



KERNELIZED LOCALITY SENSITIVE HASHING
FOR
FAST IMAGE LANDMARK ASSOCIATION

THESIS

Mark Weems, Captain, USAF

AFIT/GE/ENG/11-40

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

KERNELIZED LOCALITY SENSITIVE HASHING
FOR
FAST IMAGE LANDMARK ASSOCIATION

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Electrical Engineering

Mark Weems, B.S.E.E.
Captain, USAF

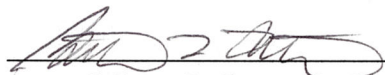
March 2011

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

KERNELIZED LOCALITY SENSITIVE HASHING
FOR
FAST IMAGE LANDMARK ASSOCIATION


Mark Weems, B.S.E.E.
Captain, USAF

Approved:



Gilbert L. Peterson, PhD (Chairman)

17 MAR 2011
date



Maj. Kenneth A. Fisher (Member)

17 March 2011
date



Dr. John F. Raquet (Member)

18 MAR 2011
date

Abstract

As the concept of war has evolved, navigation in urban environments where GPS may be degraded is increasingly becoming more important. Two existing solutions are vision-aided navigation and vision-based Simultaneous Localization and Mapping (SLAM). The problem, however, is that vision-based navigation techniques can require excessive amounts of memory and increased computational complexity resulting in a decrease in speed. This research focuses on techniques to improve such issues by speeding up and optimizing the data association process in vision-based SLAM. Specifically, this work studies the current methods that algorithms use to associate a current robot pose to that of one previously seen and introduce another method to the image mapping arena for comparison. The current method, *kd*-trees, is efficient in lower dimensions, but does not narrow the search space enough in higher dimensional datasets. In this research, Kernelized Locality-Sensitive Hashing (KLSH) is implemented to conduct the aforementioned pose associations. Results on KLSH shows that fewer image comparisons are required for location identification than that of other methods. This work can then be extended into a vision-SLAM implementation to subsequently produce a map.

Acknowledgements

First and foremost, I would like to thank God for helping me through this ultimate obstacle. It has been a long and hard journey and I have my wife, parents, family and friends to thank. Your support, well wishes, and general understanding for my absence in your lives this past year is greatly appreciated.

Mark Weems

Table of Contents

| | Page |
|---|-------|
| Abstract | iv |
| Acknowledgements | v |
| List of Figures | viii |
| List of Tables | x |
| List of Symbols | xi |
| List of Abbreviations | xii |
| I. Introduction | 1 |
| 1.1 Overview | 2 |
| 1.2 The Data Association Problem | 3 |
| 1.2.1 Exact Match vs k -Nearest Neighbors | 4 |
| 1.2.2 k -d trees | 4 |
| 1.2.3 Kernelized Locality-Sensitive Hashing | 5 |
| 1.3 Goals | 5 |
| 1.4 Results | 5 |
| 1.5 Thesis Outline | 6 |
| II. Related Work | 7 |
| 2.1 SLAM Methods | 8 |
| 2.1.1 Extended Kalman Filter | 8 |
| 2.1.2 Particle Filter | 10 |
| 2.1.3 Rao-Blackwellized Particle Filter | 12 |
| 2.1.4 FastSLAM | 13 |
| 2.2 Feature Extraction | 14 |
| 2.2.1 Interest Point Detectors | 15 |
| 2.3 Data Storage & The Data Association Problem | 15 |
| 2.3.1 Data Association in SLAM | 15 |
| 2.3.2 Linear Search | 16 |
| 2.3.3 Nearest Neighbor Search Methods | 16 |
| 2.3.4 kd -Trees | 17 |
| 2.3.5 LSH | 20 |
| 2.4 Exact Match Search via Line Fitting | 23 |
| 2.4.1 Least Squares Estimation | 24 |
| 2.4.2 Least Median Squares | 26 |
| 2.5 Summary | 26 |

| | Page |
|--|------|
| III. Methodology | 27 |
| 3.1 Implementation Background | 27 |
| 3.1.1 Feature Extraction | 27 |
| 3.1.2 Kernelized Locality-Sensitive Hashing | 32 |
| 3.1.3 RANSAC | 39 |
| 3.1.4 Integration into Data Association Implementation | 42 |
| 3.2 Computing the Data Association | 43 |
| 3.2.1 An Exact Match From k -NNs | 46 |
| 3.3 Summary | 47 |
| IV. Results & Analysis | 48 |
| 4.1 Testing Procedure | 48 |
| 4.1.1 Terrain | 48 |
| 4.1.2 Agent | 48 |
| 4.1.3 Image and Sensor Data Retrieval | 50 |
| 4.2 Implementation Details & Parameters | 51 |
| 4.2.1 Memory Obstacles | 51 |
| 4.2.2 Kernel Matrix Generation | 52 |
| 4.3 Results | 53 |
| 4.3.1 KLSH parameters | 54 |
| 4.3.2 RANSAC Parameter, R | 57 |
| 4.3.3 SIFT vs HOG Features | 58 |
| 4.3.4 Best Choice of NN | 59 |
| 4.3.5 Data Association Performance | 60 |
| 4.3.6 Flaws in Accuracy | 60 |
| 4.4 Summary | 60 |
| V. Conclusion & Future Work | 64 |
| 5.1 Conclusions | 64 |
| 5.1.1 KLSH Performance | 64 |
| 5.1.2 SIFT vs HOG | 64 |
| 5.1.3 Evaluation | 65 |
| 5.2 Future Work | 65 |
| Appendix A. Supplementary Math | 67 |
| A.1 Hamming Distance Metric | 67 |
| A.2 Central Limit Theorem | 67 |
| A.3 2D Homography Estimation | 68 |
| Bibliography | 72 |
| Author Index | 1 |

List of Figures

| Figure | | Page |
|--------|--|------|
| 2.1. | EKF Algorithm | 9 |
| 2.2. | kd -tree Example | 19 |
| 2.3. | cR -Near Neighbors Illustration | 21 |
| 2.4. | LSH Random Projection Example | 23 |
| 2.5. | LSH Algorithm | 24 |
| 2.6. | Least Squares Line Fitting Example | 25 |
| 2.7. | Least Squares Estimation Failure Example | 25 |
| 3.1. | Implementation Flow Block Diagram | 28 |
| 3.2. | HOG Implementation Flow Block Diagram | 30 |
| 3.3. | HOG Concatenation Assignment | 32 |
| 3.4. | KLSH Projection Example | 34 |
| 3.5. | KLSH Algorithm | 39 |
| 3.6. | Inlier/Outlier Identification | 40 |
| 3.7. | RANSAC Distance Threshold Effects | 42 |
| 3.8. | Data Association Computation Example: Sample Path | 44 |
| 3.9. | Data Association Computation Example: KLSH Input | 45 |
| 4.1. | Environment Layout. | 49 |
| 4.2. | Pioneer 2-AT Hardware | 49 |
| 4.3. | Pioneer 2-AT Vehicle Used for Data Collection | 50 |
| 4.4. | Parameter Sweep: KLSH n, t, b | 56 |
| 4.5. | Parameter Sweep: RANSAC Iterations | 57 |
| 4.6. | Implementation Performance: SIFT vs HOG | 58 |
| 4.7. | Implementation Performance: Probability of Accurate Match with Varying Number of HOG Features | 59 |
| 4.8. | Parameter Sweep: KLSH k | 62 |

| Figure | | Page |
|--------|---|------|
| 4.9. | Implementation Performance: Gaussian vs Chi-Squared Kernel Comparison | 63 |
| A.1. | Homography Example | 68 |
| A.2. | Mosaicking Example | 69 |

List of Tables

| Table | | Page |
|-------|---|------|
| 3.1. | KLSH Hashing Scheme | 35 |
| 3.2. | Data Association Computation Example: KLSH Output | 45 |
| 3.3. | Data Association Computation Example: Identify Image Representation | 46 |
| 3.4. | Data Association Computation Example: Normalize Representation | 46 |
| 3.5. | Data Association Computation Example: Query k -NNs | 46 |
| 4.1. | Database Reduction Heuristic: Orientation Classification . . . | 52 |

List of Symbols

| Symbol | | Page |
|--------------------|--|------|
| k -NN | Number of Nearest Neighbors | 16 |
| p | A point in \Re^d space. Represents a feature vector | 22 |
| q | A query point in \Re^d space. Represents a feature vector . . | 22 |
| \mathcal{H} | Hash Function Family | 33 |
| $h(p)$ | Hash Function | 33 |
| \vec{r} | Random Hyperplane Used to Form Hash Function | 33 |
| \mathcal{N} | Normal (Gaussian) Distribution | 33 |
| $g(p)$ | Hash Function Bucket | 34 |
| \mathcal{P} | \Re^2 Space Containing Points p_1, \dots, p_N | 34 |
| b | Number of Hash Tables | 35 |
| $\kappa(x_i, x_j)$ | The Mapping of Points x_i and x_j to Kernel Space | 35 |
| $\phi(x)$ | Kernel Space Representation of Point x | 36 |
| t | Number of Kernel Objects Used to Estimate z_t | 36 |
| \tilde{z}_t | Random Vector Used to Form KLSH Hash Function | 36 |
| n | Number of Database Objects Used to Form Hash Function | 36 |
| \vec{r} | Random Vector Used to Form Hash Function | 38 |
| H | Homography Transformation Matrix | 43 |
| R | Number of RANSAC Iterations | 43 |
| $N_{F,\theta}$ | Number of DF Corresponding to the Orientation of the Current Query | 52 |
| ι | Number of KLSH iterations | 52 |

List of Abbreviations

| Abbreviation | | Page |
|--------------|--|------|
| UAV | Unmanned Aerial Vehicle | 1 |
| UGV | Unmanned Ground Vehicle | 1 |
| ISR | Intelligence, Reconnaissance, and Surveillance | 1 |
| VAN | Vision-Aided Navigation | 1 |
| SLAM | Simultaneous Localization and Mapping | 2 |
| EKF | Extended Kalman Filter | 8 |
| RBPF | Rao-Blackwellized Particle Filter | 12 |
| NN | Nearest Neighbor | 16 |
| ANN | Approximate k-Nearest Neighbors | 16 |
| BBF | Best Bin First | 18 |
| LSH | Locality-Sensitive Hashing | 20 |
| LSE | Least Squares Estimation | 24 |
| LMS | Least Median Squares | 26 |
| DOG | difference of Gaussians | 29 |
| HOG | Histograms of Oriented Gaussians | 29 |
| KLSH | Kernelized Locality-Sensitive Hashing | 32 |
| RANSAC | Random Sampling and Consensus | 39 |
| MSS | Minimum Sample Set | 40 |
| CS | Consensus Set | 40 |
| IID | Image ID | 45 |
| QF | Query Feature | 45 |
| DF | Database Feature | 45 |
| RBF | Radial Basis Function | 52 |

KERNELIZED LOCALITY SENSITIVE HASHING FOR FAST IMAGE LANDMARK ASSOCIATION

I. Introduction

Current military and commercial operations already implement some level of autonomy in today's systems [61]. From unmanned aerial vehicles (UAV) to unmanned ground vehicles (UGV), great successes have been made to remotely carry out missions. UAVs are used today in military combat missions for intelligence, reconnaissance and surveillance (ISR) and even close air support and air interdiction as demonstrated by the RQ-1 Predator. UGVs, such as the iRobot 510 Packbot, provide military ground troops with reconnaissance, surveillance, and first responder capabilities such as explosive and other hazardous material detection and disposal.

Increasing the autonomy in these types of vehicles as well as others requires computer vision [60]. This comes in many forms but overall narrows down to the ability to use sensors to gather details about the environment and make decisions based off of those details without human interaction. For instance, in object detection, a sensor may detect an object such as a tree. The robot will use this information to decide whether it can pass through obstruction, over or around it. The Boston Dynamics LS3 scheduled for deployment in 2012 is one such robot having the expected capability to follow a leader or travel to designated locations with the aid of GPS all while carrying large loads over long distances [26]. To obtain and improve on large scale implementations such as this, autonomous navigation and computer vision through the use of images is key [60]. On a much smaller scale, research has been completed using computer vision to conduct tasks such as vision-aided navigation (VAN) [56], [68], [79]. One VAN subset in particular, robotic mapping, has made

great strides since first developed in the late 1980s [69]. Entire environments can be mapped from probabilistic approaches using many different types of sensors.

The concept of robotic mapping starts with the localization problem, which includes position tracking, global localization and solving the kidnapped robot problem [41], [77]. Localization requires the use of sensors to match details that a robot currently sees with details that are known about the environment. From there, the theory progresses to the ability to create a map of an environment without any prior knowledge. This thesis focuses on techniques to aid the mapping process through the use of images.

1.1 Overview

Consider an agent in an unknown environment. As it randomly wanders, each instance is an observation, and each observation an input to an overall map. While these observations and maps take many forms which will be discussed throughout this thesis, the point is that the recognition of observations help an agent to learn the boundaries and obstacles of this unknown environment throughout exploration and to localize itself. Those three concepts sum up the entire environment learning process: exploration, localization, and mapping [44]. Exploring an environment while using observations to localize and create a map is known as dead reckoning. This involves using observations such as odometry to calculate change in location from its original starting position. This process has no memory and therefore cannot recognize repeated terrain. This concept of recognizing previously traveled terrain is known as Simultaneous Localization and Mapping (SLAM) [25], [58]. However, the problem in image recognition or more specifically, re-recognition is that objects in an environment (and observations in general) are viewed differently depending on lighting, orientation, and overall position compared to the last time that object was viewed. Therefore, if a scene in an environment that was captured previously is recognized as new, the map reflects this as uncharted territory. In order to conduct SLAM, the agent needs to be able to efficiently match, or re-recognize observations to a database of observations.

This is known as the data association problem and is discussed in detail throughout this thesis.

There are many types of observations agents collect such as vehicle odometry information, inertial measurements, range information from lasers and sonar, and images. This thesis focuses on images, inertial measurements and odometry. The images are used to conduct the data association, while all three can be used to subsequently produce a map.

1.2 The Data Association Problem

The data association problem is best explained by comparing human to computer vision. Given two images of a particular scene, a human identifies details of what is specifically in the images and how they compare in terms of objects, people, places, structures, etc. However, computer vision uses features based on contrast, orientation, scale, etc. Therefore a human may describe an image as being a house with a car in the driveway while a computer would describe a particular point on the edge of the roof at which a drastic contrast change occurs between the roof and the sky. This description is known as a feature.

In comparing images, a human would see that the same objects in the first image are seen in the second and therefore are of the same scene. A computer calculates the spatial distance between similar features in the images. If there are a lot of feature matches between the images, then there is a high probability that the two images are of the same scene.

Computer vision methods therefore must compare every feature in an image to every other feature that the agent has discovered thus far to determine whether the location is new or previously seen. This however requires every feature discovered to be stored. Feature storage for data association can require large amounts of disk space and memory; and methods to organize them can be computationally expensive. These methods typically take on the form of data structures such as trees or hash tables.

Subsequently, the correspondence piece of data association can be computationally expensive as well, because finding an exact match out of many requires analysis of every entry in the database.

1.2.1 Exact Match vs k -Nearest Neighbors. Sometimes, the most computationally efficient solution instead of finding an exact match is to find a few close ones. These are known as nearest neighbors to the query [52]. The idea is that there should be a high probability that the exact match is among these neighbors. As alluded to earlier, matches among features are made by spatial distance calculations. By visualizing each feature as some d -dimensional point in \mathbb{R}^d space, the distance between two points corresponds to their spatial distance. Common distance metrics in image mapping are Euclidean and Mahalanobis distance. Given a query feature, q , and a set of database features, $\{p_1, \dots, p_N\}$, an exact match search would find the point, p such that the distance between the query and the match is below a threshold and closer than all other points. This, however, does not guarantee a correct match; therefore, nearest neighbor searches are implemented to identify the closest k distances to the query. The best choice of k to ensure the correct match is found is problem dependent.

1.2.2 k -d trees. Currently the most common method to conduct data association for image features is through the use of k -d trees [66], [5], [35], [13]. k -d trees are a binary tree search method of conducting the data association. The variable k in this instance, however, refers to the dimensionality of the data rather than the number of nearest neighbors. These have been shown to produce successful data associations in databases with hundreds of thousands of image features. The problem with trees however, is that they require re-balancing and re-pruning as they grow larger which can be computationally expensive. They are built with complexity $O(N)$ in which N is the time and complexity it takes to complete 1 operation. Tree maintenance is then conducted with $O(N \log N)$ [52]. As the number of features continuously grows, they can also become overcrowded which adversely affects the data association [66].

1.2.3 Kernelized Locality-Sensitive Hashing. Kernelized Locality-Sensitive Hashing (KLSH) [42] is a hash table-based method of conducting the data association. An extension of Locality-Sensitive Hashing, KLSH is a robust method that conducts the nearest neighbor search of a query by associating matching or near-matching features to the same location in a lookup table. Furthermore, differing from its parent methods, it calculates all of the hashes in kernel space. This method has been used previously on nearest neighbor search of images in many dimensions from Internet datasets such as Flickr as well as others [42]. This research analyzes the robustness of this method in the extension of its use in data association for image mapping purposes.

1.3 Goals

This research focuses on improving the data association techniques needed for successful mapping in vision-SLAM through the use of KLSH. The goals of this research are to:

- Successfully implement KLSH as a data association technique for use in vision-based SLAM
- Determine the metric position estimate as a result of the data association output
- Compare the correspondence accuracy of commonly used SIFT features vs HOG features
- Determine the best parameter set for the KLSH algorithm
- Determine whether the best choice of the number of nearest neighbors is dependent upon the overall size of the dataset

1.4 Results

The contributions of this research are the test results which provide an analysis of the robustness of using KLSH to accomplish the data association in image mapping

implementations such as vision-based SLAM and the recommendations of KLSH parameter settings for use in SLAM. The following provides the areas of focus in which the method will be evaluated on:

- Recommend the best choices for the number of neighbors, k , required to successfully associate a query feature to a matching feature in an entire database
- Evaluate performance in the context of probability of accurate match
- Evaluate performance in the context of memory and storage requirements

1.5 Thesis Outline

Chapter II first discusses vision-SLAM algorithms, then details other methods to conduct each stage of the SLAM process. Then the background information for this implementation is explained. Chapter III details how the data association implementation was conducted, while Chapter IV reports implementation test results. Finally, future work is discussed in Chapter V.

II. Related Work

While in the past SLAM and other mapping algorithms have been implemented using a variety of sensor inputs to include range information from lasers [27], [28], [58] and sonar [44], this thesis explores methods using images. Vision-SLAM is not as widely used as other methods because of the complexity required in image processing [16]. The memory required to store features and the time required for processing typically prohibits online capability or requires trade-offs in accuracy. Vision-SLAM algorithms vary in the types of additional sensors used, number and placement of cameras and how the pose estimation is refined to create a map.

Many SLAM implementations use other sensor inputs to calculate pose estimation using the images for verification and weighting. However, research has been done using vision as the only sensor in conjunction with odometry such as [9] (only camera was used for localization), [22], [35], [40], [48], [67], [78]. Stereo implementations typically use either epipolar geometry to match feature movement between cameras or visual geometry to calculate interframe movement [10], [13], [18], [64]. Other implementations use multiple cameras to obtain a 360° view with omni-directional sensing [39] as well as use features from hallway ceilings [36]. As the quality of features that can be pulled depends on the environment, features extracted from outdoor environments, open and monotonous areas in particular are generally not the strongest in tracking and are sparse. Therefore most vision-based SLAM research has been conducted indoors, but outdoor implementations have been implemented as well [5], [48]. In addition to the solving the data association problem, there also exists the loop closing problem. This is the issue of determining whether a feature is new or previously seen needing association. Both [11] and [57], via different methods, implement loop closure techniques through the SLAM system learning and classifying the appearance of all the features seen thus far as one subset. The sections that follow discuss common methods to derive the final map through pose estimation.

2.1 SLAM Methods

2.1.1 Extended Kalman Filter. The (EKF), per the name, is an extension of the Kalman filter but allows for non-linear assumptions to be made in modeling. As with similar filters to be discussed in this section, they all begin with the state vector, x_t . Shown below, the 2D state vector consists of robot pose and maps at time, t

$$x_t = (s_t, m_K)^T \quad (2.1)$$

$$x_t = (s_{x,t}, s_{y,t}, s_{\theta,t}, m_{1_{x,t}}, m_{1_{y,t}}, m_{2_{x,t}}, m_{2_{y,t}}, \dots, m_{K_{x,t}}, m_{K_{y,t}})^T$$

These poses, s_t consist of all x and y robot locations at each time instance as well as the orientation, θ . The map, m_K , at each time instance consists of the coordinate locations all of the features or landmarks observed at the corresponding robot pose, while K denotes the number of features in the map. Therefore the each state vector is a $2K + 3$ length vector. This example denotes the state vector for 2D position and estimation, however, this expression can be extended to higher dimensions.

To calculate and track the pose, the Kalman filter is basically a Bayesian filter that represents each posterior as a jointly Gaussian distribution given by

$$p(x_t | z_t, u_t) = p(s_t, m_K | z_t, u_t) \quad (2.2)$$

assuming x_t is a Gauss-Markov process. This states that the jointly Gaussian representation of the robot pose and map is represented by parameters u_t of size $2K + 3$ dimensions and the covariance matrix, Σ_t of size $(2K + 3) \times (2K + 3)$. z_t represents the sensor data providing the odometry information used to re-estimate robot location. The problem with the Kalman filter lies within its assumptions. The key assumption is that the motion model, or function that estimates the state vector at time t , is linearly dependent upon the previous pose and map at time $t - 1$ with added Gaussian noise. As with most state transitions and models, this linearity relationship isn't

common. Typically, a robot such as described in this paper moves in a circular trajectory [70] resulting in a non-linear pose function. Additionally, some sensors may be appropriately modeled as non-linear functions of the pose as well. To resolve this, the model is approximated using the Taylor Series expansion [69]. This approximation is the driving force in the extended Kalman filter relaxing the linearity assumption. This allows the motion model of transitions, x_t between time $t - 1$ and t as well as measurements, z_t to be characterized as:

$$\begin{aligned} x_t &= f(x_{(t-1)}, u_t) + w_t \\ z_t &= h(x_t, m_K) + v_t \end{aligned} \quad (2.3)$$

in which f is the motion model using previous states, $x_{(t-1)}$ and input controls, u_t to estimate the current state, x_t , and h is the observation model of each observed landmark, m_K , at pose x_t . w_t and v_t represent the zero-mean uncorrelated Gaussian noise accounting for the unmodeled errors within the motion model and observations, respectively. The general method for the EKF algorithm, as applied to SLAM, is shown in Fig. 2.1

EKF Algorithm

1. Obtain initial state estimate using odometry data
2. Observe and add landmarks to the state
3. After moving, update the new current state using the odometry data
4. Update the estimated state from re-observing the landmarks
5. Add new landmarks to the current state

Figure 2.1: **EKF Algorithm.**

Each estimate step results in a calculation of the state vector consisting of robot pose and observed map landmarks as well as a covariance matrix P representing the following covariances [58]:

The update step then yields the following EKF equations as described by [54]

$$P_{t|t} = \begin{bmatrix} P_{xx} & P_{xm} \\ P_{xm} & P_{mm} \end{bmatrix}_{t|t} \quad (2.4)$$

$$= \begin{array}{|c|c|c|c|c|c|} \hline & & \dots & \dots & \dots & \dots \\ \hline \text{\textit{Cov}}(s_{x,t}, s_{y,t}, s_{\theta,t})_{3 \times 3} & \text{\textit{Cov}}(m_1, s_t) & \dots & \dots & \dots & \dots \\ \hline & & \dots & \dots & \dots & \dots \\ \hline & & \dots & \dots & \dots & \dots \\ \hline \text{\textit{Cov}}(s_t, m_1) & \text{\textit{Cov}}(m_{1_{x,t}}, m_{1_{y,t}})_{2 \times 2} & \dots & \dots & \text{\textit{Cov}}(m_{1_t}, m_{K_t})_{2 \times 2} & \\ \hline & & \dots & \dots & & \\ \hline \dots & \dots & \dots & \dots & \dots & \dots \\ \hline \dots & \dots & \dots & \dots & \dots & \dots \\ \hline & & & & & \\ \hline & \text{\textit{Cov}}(m_{K_t}, m_{1_t})_{2 \times 2} & & & \text{\textit{Cov}}(m_{K_{x,t}}, m_{K_{y,t}})_{2 \times 2} & \\ \hline & & & & & \\ \hline \end{array}$$

$$\begin{bmatrix} \bar{x}_{t|t} \\ \bar{m}_t \end{bmatrix} = [\bar{x}_{t|t-1} \ \bar{m}_{t-1}]^T + W_t [z_t - h(\bar{x}_{t|t-1}, \bar{m}_{t-1})] \quad (2.5)$$

$$P_{t|t} = P_{t|t-1} - W_t S_t W_t^T$$

$$S_t = \nabla h P_{t|t} \nabla h^T + R_t$$

$$W_t = P_{t|t-1} \nabla h^T S_t^{-1}$$

where ∇h is the Jacobian of h calculated at $\bar{x}_{t|t-1}$ and \bar{m}_{t-1} . $[z_t - h(\bar{x}_{t|t-1}, \bar{m}_{t-1})]$ is called the innovation and is defined as the difference between the estimation and the actual observation. S_t is then defined as the innovation covariance while W_t is the Kalman gain. The Kalman gain is a weighting applied to update calculations that directly affects how much each observed landmark should be trusted in relation to the error calculations that are made.

2.1.2 Particle Filter. Particle filters represent a distribution by sampling from that distribution. They can now be considered approximate, nonparametric and represent any distribution, even Gaussian if needed. First we denote the particles, which are the samples from the posterior distribution or more specifically, the state vectors:

$$\chi_t = x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, \dots, x_t^{[M]}. \quad (2.6)$$

Here M is the number of particles in the set χ_t . Typically this number is large (i.e., tens of thousands). The likelihood that a particular state hypothesis, x_t to be among the particle set can be found using its Bayes filter relationship:

$$x_t^{[m]} p(x_t | z_{1:t}, u_{1:t}). \quad (2.7)$$

[70] explains that this relationship remains true theoretically as $M \rightarrow \infty$, but $M \geq 100$ is typically suitable in practice. At each time t , a new sample hypothesis is generated for each m^{th} particle from the distribution $x_t^{[m]} p(x_t^{[m]} | u_t, x_{t-1}^{[m]})$, based on the previous particle $x_{t-1}^{[m]}$ as well as the control used, u_t .

Next the importance factor or particle weight, $w_t^{[m]}$ is found.

$$w_t^{[m]} = p(z_t | x_t^{[m]}) \quad (2.8)$$

By examining this likelihood, the weight represents the possibility that the m^{th} particle is the correct representation of the environment considering the observed measurements, z_t and the known statistics, $x_t^{[m]}$. Next the particles are resampled. A new temporary particle set, $\bar{\chi}_t$ is formed by randomly drawing with replacement M particles. The probability of drawing the m^{th} particle is a function of its weight, $w_t^{[m]}$.

By drawing M random uniformly distributed numbers on interval $[0, 1]$ and selecting the particle that corresponds to the weight range of that random number, a new particle set will be formed. For each m , the old particle will be replaced with the drawn particle and reassigned a weight of $\frac{1}{M}$. Ideally, since this step is conducted with replacement, the particles with higher importance or stronger weights tend to be chosen more while the particles of less importance are eliminated.

2.1.3 Rao-Blackwellized Particle Filter. The Rao-Blackwellized Particle Filter RBPF is a Monte Carlo filtering technique. The drawback of the particle filter is that in high dimensional spaces, sampling can be inefficient. Rao-Blackwellisation is the recognition that the model is tractable and can be analytically marginalized out or represented with only the necessary segments of the data. This leads directly in to the advantage. In following this strategy, the size of the space requiring sampling is greatly reduced.

Like ordinary particle filters, the state space is generalized as

$$p(y_{0:t}|z_{1:t}) = p(y_{0:t-1}|z_{1:t-1}) \frac{p(z_t|y_t)p(y_t|y_{t-1})}{p(z_t|z_{1:t-1})} \quad (2.9)$$

where y_t and z_t are the state vectors and observations respectively. The fundamental piece in the Rao-Blackwellisation is that the state space is divided into two groups, r_t and x_t resulting in

$$p(y_t|y_{t-1}) = p(x_t|r_{t-1:t}, x_{t-1})p(r_t|r_{t-1}). \quad (2.10)$$

This allows $x_{0:t}$ to be marginalized out of the conditional posterior distribution as shown in Equation (2.11)

$$p(x_{0:t}y_{1:t}, r_{0:t}) \rightarrow p(r_{0:t}|z_{1:t}), \quad (2.11)$$

which is just a fraction of the entire sample space. The importance sampling then becomes

$$(\bar{r}_t^m) q(r_t|r_{0:t-1}^m, z_{1:t}) \quad (2.12)$$

with weights

$$w_t^m = \frac{p(\bar{r}_{0:t}^m|z_{1:t})}{q(\bar{r}_t^m|\bar{r}_{0:t-1}^m, z_{1:t})p(\bar{r}_{0:t-1}^m|z_{1:t-1})} \quad (2.13)$$

and normalized by

$$\tilde{w}_t^m = w_t^m \left[\sum_{j=1}^N w_t^j \right]^{-1} \quad (2.14)$$

In selection the samples, $\tilde{r}_{0:t}^m$ are multiplied by large importance weights and suppressed by smaller weights to obtain M random samples, $\tilde{r}_{0:t}^m$. This distribution is approximated as $p(\tilde{r}_{0:t}^m | y_{1:t})$. Finally, apply a Markov transition kernel to obtain $r_{0:t}^m$ given by invariant distribution $p(r_{0:t}^m | y_{1:t})$.

2.1.4 FastSLAM. FastSLAM differs from other SLAM algorithms in that it infers that the posterior can be factored [53].

$$p(\theta, s^t | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_n p(\theta_n | s^t, z^t, u^t, n^t) \quad (2.15)$$

It samples the path using a particle filter, in which each particle m , is its own version of the map. But each particle also consists of N extended Kalman filters. The m^{th} particle contains the following EKF parameters:

$$S_t^{[m]} = s^{t,[m]}, u_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, \dots, u_{N,t}^{[m]}, \Sigma_{N,t}^{[m]} \quad (2.16)$$

The original FastSLAM, [50] makes the pose and observation estimates independent of each other measuring the pose through a proposal distribution based solely on the recent motion command and the observations through resampling. In situations of high motion noise, this leads to the sampling of unlikely poses. FastSLAM 2.0, [51] samples the pose as a function of both the control and the measurements yielding

$$s_t^{[m]} \approx p(s_t | s^{t-1,[m]}, u^t, z^t, n^t). \quad (2.17)$$

This formulates the factorable proposal distribution as

$$p(s_t | s^{t-1,[m]}, u^t, z^t, n^t) = \eta^{[m]} \int p(z_t | \theta_{n,t}, n_t, s_t) p(\theta_{n,t} | s^{t-1,[m]} z^{t-1} n^{t-1}) \quad (2.18)$$

Then resampling is conducted with

$$w_t^{[m]} \propto p(z_t | s^{t-1,[m]}, u^t, z^t, n^t) = \int \int p(z_t | \theta_{n,t}, s_t, n_t). \quad (2.19)$$

Finally, feature tracking is ensured reliable by the presence and absence of features in evidence calculations. Originally from [24], this algorithm is modified and ensures likely feature tracking through log-odds of the physical existence of landmarks calculations.

2.2 *Feature Extraction*

No matter what mapping filter method is used, vision SLAM and the image mapping arena as a whole depends on the ability to retrieve mathematically interesting keypoints in an image. More importantly those same keypoints need to be recognized in similar images. When a person looks at an image, key features would be distinct objects, buildings or interesting scenic points that would strike memory upon recognition in another image. However, in computer vision features are typically detected by distinctive lighting, intensity, orientation changes as well as expected geometric and photometric deformations throughout the image. Furthermore, features that are found in an image need to be invariant against these as well translation, rotation, and scale changes. How much invariance is required, however, is dependent upon the problem at hand [6]. Many researchers have found strong techniques that have shown invariance in many aspects of images and environments both indoor and outdoor. More well known techniques have been finding dissimilarity and texture [65], using Hessian matrices [6], multi-scale phase based features [14], as well as one evaluated later in this thesis by called Scale Invariant Feature Transforms (SIFT) [46].

SIFT has been very successful in the computer vision community and frequently used in vision aided navigation [75] and image mapping [5], [9], [22], [18].

2.2.1 Interest Point Detectors. In order to help find these interesting points in an image, interest point detectors have been implemented. These help to narrow down the image space in which feature detection methods are used. Harris Corners [33], developed in 1988 are a combination of scale-variant corner and edge detectors. This method provided a basis for feature extraction and is therefore followed by many variations. Region detectors such as the affine and scale invariant salient regions found by [37], [38] are also popular. Finally blobs or regions lighter than their background defined by smoothed curved edges, have been used as well [21].

2.3 Data Storage & The Data Association Problem

The data association problem is probably the most crucial part in image mapping. A wrong association will cause filter methods to diverge and maps to fall apart. The problem is that direct search methods such as linear search can be computationally expensive. And while distance metrics identify matches well, determining outliers or mismatches can be even more expensive. These are the instances in which it is better to find a few database objects that are close in distance to a query rather than an exact match. These nearest neighbors have a high probability of match. Given a query feature, q and a set of database features, $\{p_1, \dots, p_N\}$ an exact match would be such that the distance between the query and the match is below a set threshold and closer than all other points. Finding the k -nearest neighbors entails identifying the closest k distances to the query. In covering common data association methods, this section reviews two common types of data structures used to store data for improved search techniques, trees and hash tables.

2.3.1 Data Association in SLAM. While the FastSLAM and FastSLAM 2.0 implementations both used maximum likelihood [49], [51] to solve the data association problem, the most common method has been to store and match image features using

kd-trees. Both [57] and [67] use *kd*-trees with SIFT features in conjunction with SLAM via particle filtering, while [35] used them for 3D SLAM but added a Best Bin First heuristic to the *k*-d tree search. Finally, [18] used vision-SLAM to map an environment through visual odometry by associating SIFT features using hash tables.

2.3.2 Linear Search. The most basic method of solving the data association is through linear search. In the context of matching images, this is conducted by matching each feature to every other feature in the database, typically by Euclidean or Mahalanobis distance methods. While this may be sufficient for small datasets in low dimensionality, the operation time and complexity increases with $O(M^n)$ where M is the time and complexity to search 1 feature and n is the number of features in the database.

At times the most efficient option isn't necessarily to find an exact match to a specific query. Sometimes, finding a few matches similar to the exact is best. Nearest neighbor matching algorithms narrow down a large dataset to a more manageable space. Then an exact search can be conducted on those few neighbors.

2.3.3 Nearest Neighbor Search Methods . The nearest neighbor search problem states that given a set \mathcal{P} of points in \mathbb{R}^d , a data structure can be constructed which given a query point q , will find the k points in \mathcal{P} with the smallest distance to q [52]. k in this context is the number of nearest neighbors desired. As alluded to previously, distance is user defined, but typically an l_s norm is used in which

$$||p - q|| = \left(\sum_{i=1}^d |x_i|^s \right)^{\frac{1}{s}} \quad (2.20)$$

There are three main types of nearest neighbor (NN) output methods:

- Range Search
- k -NN
- Approximate k -NN (ANN)

All of these methods are based on spatial distance from a query, q to a potential matching point, p . A maximum range is established in searching for a match to a query [52]. The range is the threshold that database objects must meet in order to be matches or neighbors. There is no limit on the number of neighbors returned and there are typically no heuristics to decrease the range as closer points are found. k -nearest neighbor search simply finds the closest k points to the query. In approximate k -NN search however, there is a termination point in accordance with finding the k^{th} neighbor. Once the distance between a point, p , and q surpasses a threshold or range, no more neighbors will be returned, regardless of how far off of k .

2.3.4 kd-Trees. The kd -tree is a binary data structure tree used to store finite points from a k -dimensional space [52]. Given a set of points in \mathbb{R}^{k_d} , each can then be placed in a binary tree based on distance from q . Using search methods, the tree can prune down to a single match or a neighborhood of matches.

2.3.4.1 Tree Creation. Given an $N \times k_d$ data set in which N is the number of samples and k_d being the number of dimensions in \mathbb{R}^{k_d} space, a kd -tree can be constructed using statistics of the data as well as the d -dimensional coordinates or nodes of each sample. The leaves of the tree are nodes or sets of nodes, while the branches are splits made based on the statistics of node locations. Trees are built with $O(N)$, in which N is the time and complexity of one completing one operation.

First, the variance of the points across each dimension is analyzed. The dimension, i with the greatest variance forms the first branch split. This split is in the form of a hyperplane and is made at the median, m of the data in that dimension. The first step is repeated with the remaining dimensions; that is the split is continuously made at the median of the dimension with the greatest variance [7], [8], [47]. This ensures balance in the tree with depth, $d \leq \log_2 N$ where N is the total number of data points across all samples or in this case feature vectors.

Consider the 2D example in the left of Fig. 2.2 which forms the tree node layout shown on the right. This becomes increasingly harder to visualize as the number of dimensions increases, for example the SIFT feature vector to be discussed later is 128 dimensions in length, however the theory remains the same.

The binary tree as shown in the right of Fig. 2.2 is then built based off the splits. Each node in the tree represents a line forming a hyperrectangle in the data space. The first node represents the split made at 0.34 on dimension 1. The left and right branches then represent the points in the space to the left and right of the split, respectively. Like before, the first split on the left side occurs at the median of dimension 2 as this is the dimension with the greatest variance across the points in the hyperrectangle. This process is repeated until the branches lead to the leaves of the tree which are the data points themselves. In this tree, each leaf represents a singular data point, however some trees end in small clusters of similar data points. In the latter case each of the siblings in the leaf are the nearest neighbors of the query. Observing the tree it is seen that each node is represented by the (i, m) values denoting the dimension and value among the points the split was made.

2.3.4.2 Nearest Neighbor and Best Bin First Search. Nearest neighbor algorithms are performed finding the data point closest in Euclidean distance to the query point, or feature vector to be matched. First the tree is traversed in order to find the bin that the query point fits in according to the current tree structure. In that traversal, the one-dimensional distances to those branching points are also calculated and recorded in a priority search queue. Once the bin is found, the leaf in that bin can be tested for matching as a good approximation to the nearest neighbor. The distance, D from the q to the data point in that bin calculated. This bin is considered the best bin. All subsequent searching is completed starting with the best bin first (BBF). The tree is then backtracked, pruning off branches whose one-dimensional distances from q are greater than D . This procedure ensures the most efficient detection of the matching data point [52].

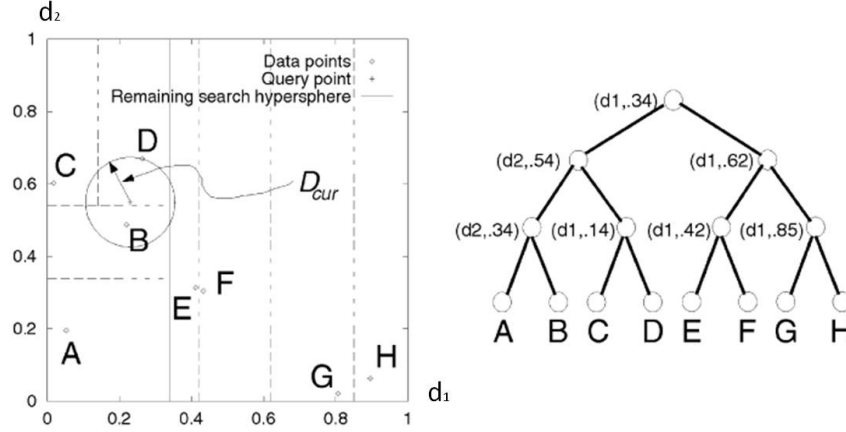


Figure 2.2: *kd-tree* example. Each of the points in \mathcal{R}^{k_d} space are shown in this $k = 2$ *kd-tree* data space illustration (left). The division lines are made based on the statistics of the points in that section. For instance, the first line dividing $A - D$ and $E - H$ was made along dimension 1 because the variance of the points across dimension 1 was greater than that of dimension 2. The line is drawn at the median of the points in dimension 1. This process is repeated until the k points are left. The binary *kd-tree* branch diagram (right) can then be formed based on the points on which side of each division line the points are on.

2.3.4.3 Tree Size. While most literature fails to comment on the size of the overall tree and number of data points or leaves that should appear at the branch ends, the understanding is that the tree should be considerably large enough so that the data space can be accurately spread throughout allowing for the smallest search space possible in trying to compare query and data feature vectors. In other words, trial and error may be the best option.

2.3.4.4 Size of k . This concept has been mainly been tested on mid-dimensional datasets of 8 – 20 dimensions [7], [8], [52]. Typically, the performance decreased as the dimensionality increased. Adding heuristics such as those noted above allowed for use in much higher data space. Similar to the intention for this work, [47] performed similar tree matching techniques using 128-dimension SIFT feature vectors with good results. The purpose of their work was to experiment with the NN algorithm using dimensionality reduction on SIFT vectors matching features

from varying viewpoints and scale changes. While they had varying results in the dimensionality reduction, they had accurate matching using the full length vector. Essentially, the rule of thumb to follow is $k \ll N$.

2.3.4.5 Tree Maintenance. Adding or deleting points from the tree can be done at any time with complexity $O(N \log N)$; however, periodic balance checks should be accomplished to ensure that the tree is still balanced accordingly. The tree shown in Fig. 2.2 is a depiction of a fully balanced tree. If nodes are deleted from one side of the tree more than the other, imbalance occurs. The degree of imbalance required to cause issues in the algorithm is dependent on the problem, but overall this balance is key when performing nearest neighbor-like algorithms.

2.3.4.6 Nearest Match vs. a close one. As remarked in [67], the disadvantage of using a kd -tree is that sometimes the match produced isn't the nearest but a close one. While some algorithms settle for this, others use heuristics, to help ensure nearest match results. One such heuristic searches through a predetermined number of branches to help ensure that the search wasn't stopped prematurely. In another implementation, [52] built multiple kd -trees, producing different branch structures while projecting points onto different hyperplanes to ensure the right data point was converged on.

2.3.5 LSH. Locality-Sensitive Hashing (LSH) was originally developed by [34] then refined by [31]. By hashing points to a series of hash tables, the algorithm ensures a high probability of collision for objects that are close to each other in distance than those that are farther apart [31]. This concept and variations thereafter ([1], [3], [15]) have been used for similarity search on several sets of data to include images in large dimensions. A new hashing family was then introduced by [2] in which the distances that defined the family were measured according to the l_s norm. Similar to the intent of this thesis, LSH was successfully used in agent localization for image mapping in [59].

A LSH algorithm is defined by the number of nearest neighbors output or how near a point should be to be considered. The first definition below describes the cR -nearest neighbor formulation. See Fig. 2.3 for an illustration. This is similar to k -NN and ANN search discussed previously.

LSH Definition 1 *Consider an $N \times d$ dataset \mathcal{P} in which N are the number of points, p and d are the number of dimensions in \mathbb{R}^d space with parameters radius, $R > 0$ and probability $\delta > 0$. A data structure can be constructed which given a query point q , which does the following with probability $1 - \delta$: in the event there is an R -near neighbor of q in \mathcal{P} , the cR -near neighbors of q in \mathcal{P} are reported.*

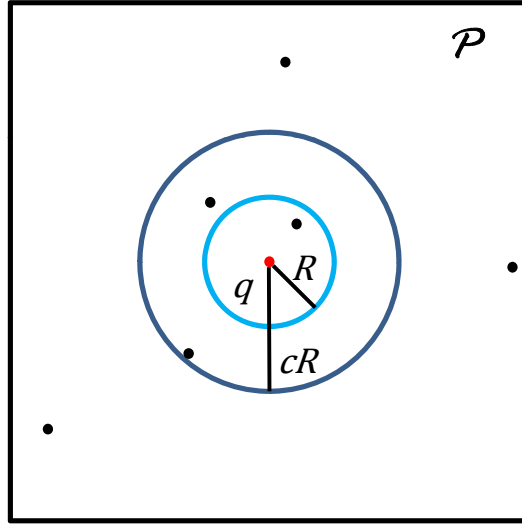


Figure 2.3: cR -near neighbors illustration from Definition 1.

LSH algorithms use more than one hash function to store data, also known as hash function families, \mathcal{H} . Increasing the number of hash functions increases the probability that a collision between p and q is much higher for points that are close and lower for points that aren't. This probability as seen in Equation (2.21) below is related to the distance between those points.

LSH Definition 2 *A family of hash functions, \mathcal{H} is called (R, cR, P_1, P_2) -sensitive if for any points $p, q \in \mathbb{R}^d$.*

$$\begin{cases} \|p - q\| \leq R & P_H[h(q) = h(p)] \geq P_1 \\ \|p - q\| \geq cR & P_H[h(q) = h(p)] \leq P_2 \end{cases} \quad (2.21)$$

P_1 and P_2 in this instance are probability thresholds that meet $P_1 > P_2$ ($R < cR$). In other words, every hash function in the family must report hash results such that when the distance between p and q is within R , there is a high probability ($\geq P_1$) of collision. Additionally, when the distance is greater than cR , there is a low probability of collision ($\leq P_2$).

As mentioned earlier, there are a few different methods to form the LSH hash tables to solve the nearest neighbor algorithm. This discussion is based on the technique developed by [31]. This technique transforms all points into the Hamming cube, H^d , to compute hashes. See Appendix A.1 for review on Hamming cube representation via the Unary function.

First, point p is represented as a binary vector, $v(p)$, with length d' , where $d' = Cd$ and C is the largest point across all dimensions in p . Next, denote I as the subset of all dimensions in d' : $I \in \{1, \dots, d'\}$. There are t hash functions in each bucket, and b buckets will represent the hash sequence for each point. One hash bucket is defined as:

$$g_j(v(p)) = v(p)_{|I_j} \quad (2.22)$$

where $v(p)_{|I_j}$ is the projection of $v(p)$ on the coordinate set I_j . The following example illustrates the computation of $v(p)_{|I_j}$ in 2 dimensions. Consider the point from Equation (A.2) in the example in Appendix A.3, $v(p) = [1110011111]$ where $d' = 10$ and set $t = 3$. At random, select t coordinates from the set, $1, \dots, d'$ to project onto, for example $I = \{2, 5, 8\}$.

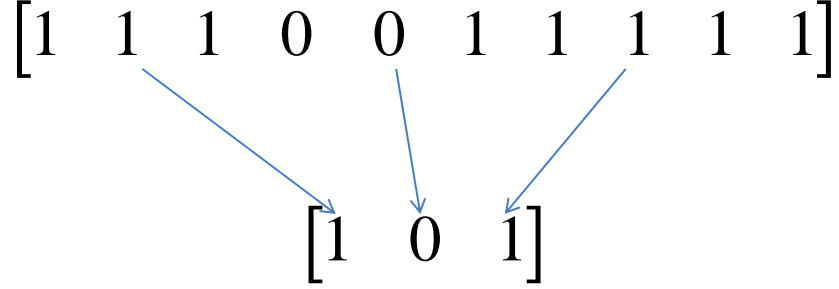


Figure 2.4: **LSH random projection example.**

Fig. 2.4 shows that the projection of the point onto the coordinate set in Hamming space is the concatenation of the bits in those positions. Thus, $g_j(p) = [1 \ 0 \ 1]$. Each bit selection in $g_j(p)$ is a different hash function, h that forms an overall hash bucket. Each bucket represents the hashing of a point to a table. This process is repeated b times such that

$$g(p) = \langle g_1(p), \dots, g_b(p) \rangle \quad (2.23)$$

These b concatenations are the indices to the b hash tables point p is stored to. Query processing simply uses the same functions to hash query points to. The act of retrieving a database point from the indices calculated is considered a collision. Points are retrieved until all b buckets have been retrieved or the total number of points retrieved exceeds $2b$. To find an exact match to q among all of the neighbors gathered, requires using any of the nearest neighbor searches mentioned in Section 2.3.3. Fig. 2.5 outlines the general algorithm.

2.4 Exact Match Search via Line Fitting

With the exception of linear search, the previous section examined data association methods to find a number similar to a given query with the high probability that the exact match is included. To then find an exact match, this section reviews line fitting techniques to conduct image registration. As image registration deals with

LSH Algorithm

1. Preprocessing
 - (a) Map all points to Hamming space.
 - (b) Choose t hash functions from \mathcal{H} to form a bucket $(g_j(p) = [h_1(p), h_2(p), \dots, h_t(p)])$
 - (c) Select b hash function buckets $(g_1(p), \dots, g_j(p), \dots, g_b(p))$
 - (d) For each j bucket, compute projection of vector, $v(p)$ on to t dimensions in the coordinate set, I_j
 - i. This forms the tuple $g_j(p) = \langle g_1(p) \dots g_L(p) \rangle$, for $1 < j < L$
2. Processing and Matching a query, q
 - (a) Hash each query point using the same b hash function buckets as in pre-processing
 - (b) Retrieve all points in the subsequent buckets $g_1(q), \dots, g_b(q)$ until either of the following occur:
 - i. All points from the b buckets have been retrieved
 - ii. Total number of points retrieved exceeds $2b$
 - (c) Use nearest neighbor search techniques to answer query

Figure 2.5: **LSH Algorithm.**

aligning images with translation, rotation, and scale changes, line fitting between many feature matches between images in the presence of outliers is key to ensuring the feature match locations in the images are the same [80].

2.4.1 Least Squares Estimation. Given a set of data points in X and Y , finding the line or model that best represents or fits the data, the most common technique is the least squares estimation LSE method. The LSE method states that out of all possible models, the best model follows the following:

$$\text{Best Model} = \min \left(\sum_{i=1}^n (y_i - f(x_i))^2 \right) \quad (2.24)$$

Fig. 2.6 shows a set of data points that have been fit to the line shown. Each residual, or the vertical distance between each point and the line must be within some

threshold to be considered an inlier. Any points beyond this threshold are considered outliers and do not fit the model that the data has been fit to.

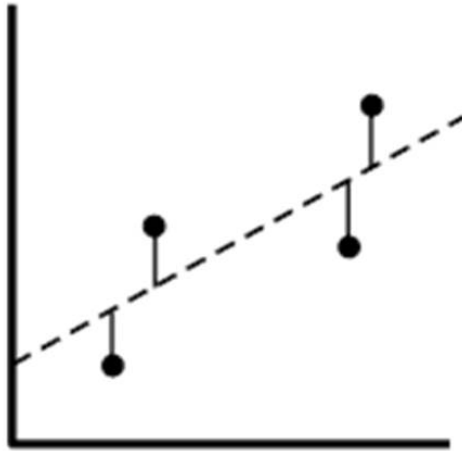


Figure 2.6: Least Squares line fitting example.

Like other techniques, least squares estimation has its flaws. At times, it can only take as little as one outlier to skew the result producing an inaccurate representation of what is constituted as an inlier. Fig. 2.7 shows an example of one such downfall.

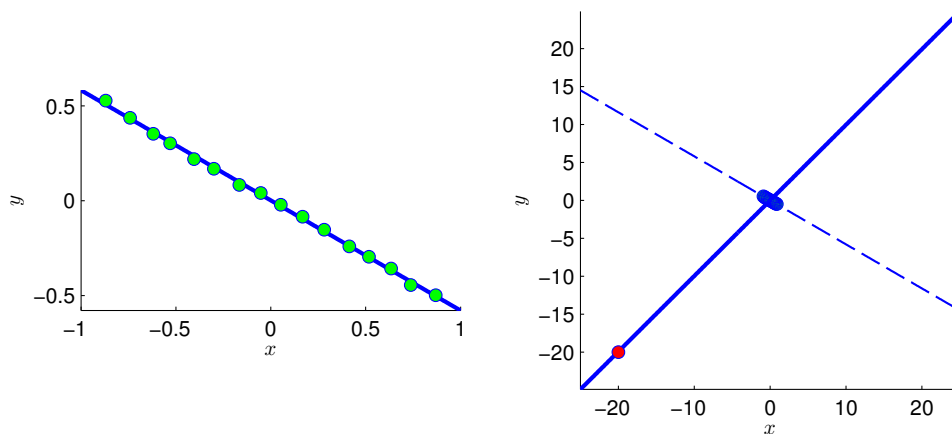


Figure 2.7: Least Squares Estimation failure example [52]. The least squares line on the left was calculated based on the points shown. The plot to the right shows the negative affect just one outlier can have on line fitting of a data set using least squares.

2.4.2 Least Median Squares. Similar to the least squares algorithm, least median squares is also widely used to model data. The least median squares (LMS) algorithm simply states, out of all possible models, choose the model that best minimizes the data such that

$$\text{Best Model} = \min (\mathbf{M}([(y_1 - f(x_1))^2, (y_2 - f(x_2))^2, \dots, (y_n - f(x_n))^2])^2) \quad (2.25)$$

where \mathbf{M} is the median of the residuals. This method is very accurate even in the presence of outliers. It outperforms other algorithms to include RANSAC, as discussed later, up to the case in which the outliers account for more than 40% of the dataset [73]. This is known as the breakdown point. This method also fails in situations in which a majority of the data is positioned on a dominant plane in the image [71]. In other words, since this algorithm is dependent upon the minimum median of the residuals, if the majority of those inputs are originating from one localized area, it can have an undesired affect on the fit line.

2.5 Summary

This chapter has provided a detailed study of all techniques required to conduct fast image retrieval for vision-based SLAM. First, SLAM filtering methods were reviewed, presenting detail on the pose estimation process. Then to answer the data association, nearest neighbor search methods using tree and hash table-based approaches were evaluated. Finally, to conduct exact match image registration to match the query based on the nearest neighbors found, common line fitting techniques were eliminated as possible options.

III. Methodology

This implementation combines the work of previous researchers and tests their capabilities in a SLAM-like application with concentration on the data association. The results of the data association testing can be used for vision-SLAM as well as other localization techniques through image mapping. The general flow is shown in Fig. 3.1. In separate tests, SIFT features and features of type Histograms of Oriented Gradients are broken into database and query features as determined by the agent's path. Each query is associated with k similar neighbors using KLSH. This method stores each feature in a hash table by transforming points into kernel space prior to completing hash calculations. Then an exact match amongst the neighbors is found using RANSAC, in which image planes are aligned by calculating a transformation matrix using a 2D image homography. Then feature matches in each image are compared using distance metrics allowing for removing of false feature matches, known as outliers. The output can then be used for localizing agent pose in the mapping process. This implementation was conducted running all landmark features through each method before proceeding to the next, but can be done iteratively. This chapter first presents the background of each method used in the implementation. It then details how these methods were used to conduct the data association for this SLAM-like setup.

3.1 *Implementation Background*

This section discusses the background and related work of the concepts implemented in this research. First, SIFT and HOG features are reviewed for image feature extraction. To compute the data association a k -nearest neighbor method, an extension of LSH, known as KLSH is discussed. Finally, RANSAC is reviewed which answers the query using the nearest neighbors provided by KLSH.

3.1.1 Feature Extraction. Using computer vision to detect features requires strong detection capability in images whose scenes have specific orientation changes among the objects and vary in scale and translation. Tracking of these features from

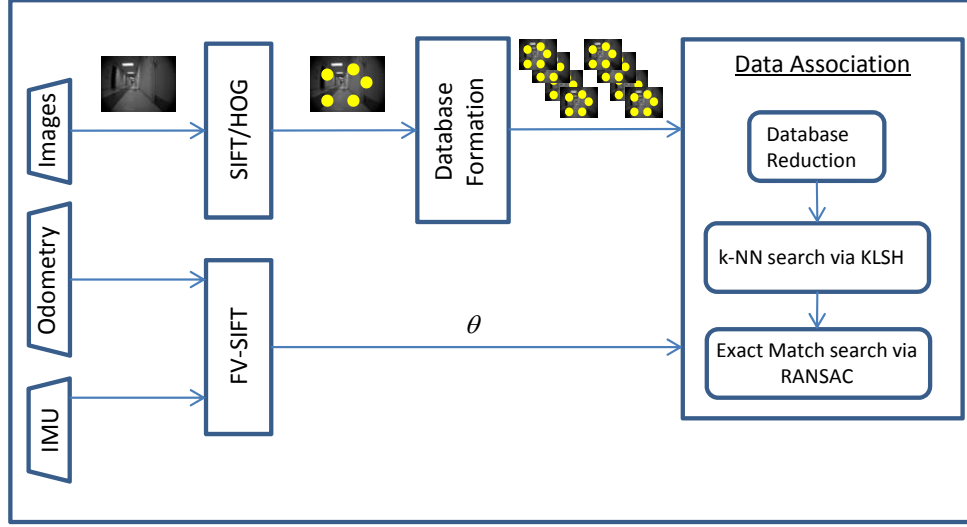


Figure 3.1: **Implementation flow block diagram.**

image to image requires repeatability of the detector so that features from the same objects or keypoints are captured. Finally features that are found in an image need to be invariant against translation, rotation, and scale changes depending on the problem. For instance, rotation invariance is required with environments in which the camera or objects being tracked are moving with changing orientations, such as images taken from a camera rotating about the horizontal axis. Scale invariance is required in scenes in which the agent is moving forward or backward through the environment, causing objects to change in size. This implementation uses an agent that moves through an indoor, land environment. Therefore, there are minimal rotation changes throughout the image, just scale and translation.

Data Association via KLSH is tested on both SIFT and HOG features. SIFT features have been found to be successful in all aspects of computer vision due to their scale, rotation, and translation invariance and have been refined on numerous occasions since originally implemented by Lowe [46]. HOG features, however are scale invariant and have mainly been used in low resolution facial and object recognition [12], [55], [20].

3.1.1.1 SIFT. Scale Invariant Feature Transforms (SIFT) have been used in many SLAM applications such as [9], [13], [35], [57], [67], [74]. First the

image is analyzed in scale space using a difference of Gaussians (DOG) function for various keypoints where interest points may be located. Local extrema of the DOG function are computed and assigned as keypoints by comparing each pixel with its surrounding 4×4 pixel neighbors. If the current pixel is greater or smaller than all of the surrounding pixels in that neighborhood it is designated a keypoint. Next, for each keypoint a Gaussian smoothing function is applied as a function of the scale at all levels in the image scale space. Next an orientation assignment is conducted using gradient calculations on a specified region around the keypoint given by:

$$\begin{aligned} m(x, y) &= \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \\ \theta(x, y) &= \tan^{-1} \left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right) \end{aligned} \quad (3.1)$$

Then all of the magnitudes are combined in a histogram, typically 36 bins long covering 360° of orientation. Peaks within this histogram are noted as dominant directions in the gradient. The orientation corresponding to the peak magnitude of the histogram is assigned as the orientation for the keypoint. Multiple keypoints and corresponding orientations are defined for all magnitudes that are within 80% of the peak. This multiple assignment is rare however greatly contributes to the stability of the feature matching algorithm. Finally, the descriptor assignment is started by applying a Gaussian blur on the region around each keypoint. Then the region is divided into 2×2 subregions. The gradient is then calculated on each subregion allowing the formation of another orientation histogram this time with typically 8 or 9 orientation bins depending on the requirements of the system. A descriptor is formed by concatenating the histograms of all subregions. The final feature is a data structure that contains information on the feature location, scale, orientation and the descriptor vector.

3.1.1.2 HOG. The Histograms of Oriented Gaussians (HOG) implementation used in this paper follows the ideas from [12], [20], [19], [55]. Previously HOG has primarily been used on small regions of an image typically for the use of

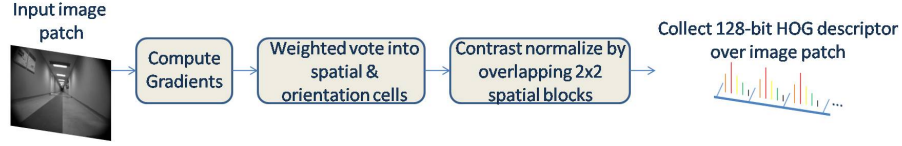


Figure 3.2: **HOG Implementation Flow Block Diagram.**

pedestrian, vehicle and object detection [4], [20], [19], [55]. It has been found to outperform many other descriptors in its class such as Haar wavelets and AdaBoost classifiers [19], [55]. The overall process is described in Fig. 3.2. In starting with an image patch, the goal is to calculate a 128-length descriptor for each patch.

First, the gradient of each image was calculated resulting in an orientation and magnitude representation for each pixel as described below.

$$\begin{aligned} \text{mag} &= \sqrt{(\nabla_x)^2 + (\nabla_y)^2} \\ \theta &= \arctan \frac{\nabla_y}{\nabla_x}, \quad (\theta \in \mathbb{R}[0, 180^\circ], \text{unsigned}) \end{aligned} \quad (3.2)$$

This gradient calculation was looked at in two ways. [20] reported that a simple 1D $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ mask worked best, but the 3×3 sobel filter tested in [20] and primarily used in [55] was also looked at for comparison. A smoothing function was not used, but instead the edges of each image gradient were set to zero to help eliminate any negative edge effects such as aliasing.

Next, a 3×3 block window was defined. While for this implementation the window had to be 3×3 to meet the requirements of the vision aided navigation algorithm being used, there are multiple combinations that work well. See [19] for a survey. Although the window block size is set, the number of pixels this window can cover is what can vary. Since in the past this method has only been used on small image regions there were not any guidelines to go by on how big or small the window should be. Therefore, the implementation was first tested with the recommendation by [20] for an 8×8 -pixel cell resulting in a 24×24 -pixel window. Each 8×8 cell is represented as a histogram with 8 orientation bins. As little as four can be used,

however better performance has been found up to nine bins (20° of separation) [20]. This implementation used eight to meet the descriptor requirements.

Although just about all feature extraction methods use gradient histograms to represent calculated features, this method differs in its specific binning tactics. Instead of just assigning the entire magnitude to an orientation bin, bin assignment is determined by a weighted vote. The two bins closest to the pixel orientation receive a percentage of the pixel magnitude based on the degree proximity from each bin. For example, a pixel orientation of 65° in a nine bin histogram would be split between bins three and four or orientations centered at 50° and 70° respectively. The weighted vote based on spatial distance would result in 75% of the magnitude being assigned to bin four (70°) and 25% assigned to bin three (50°).

Next the histograms of a 2×2 block of four cells are concatenated as a 32-bit vector ($4 \text{ cell histograms} \times 8 \text{ bins in length} = 32$). This vector is a partial descriptor for the patch and will represent the first cell in the window. The vector is then normalized to unit length by calculating the l_2 norm. This overlapping block method provides contrast invariance. By repeating this method for the other cells in the window, four cells are found to have partial descriptors comprising of their neighboring cells. An example of this is shown in Fig. 3.3. The displacement of the four copies of numbers 1 – 4 represent four partial descriptors that make up the final patch descriptor. Each respective underlined number indicates the cell that represents that respective partial descriptor. Concatenating these four partial 32-bit descriptors results in the 128-length descriptor describing this window as a feature ($32 \times 4 \text{ partial descriptors} = 128$).

As this window slides across every pixel throughout the image, thousands of feature descriptors are calculated. Obviously, since all of these descriptors aren't actually valid features, only those that meet certain standards or thresholds were kept. In any feature extraction method, one indication of a good feature is one that has distinct contrast transitions [46]. This is mathematically noticed by those

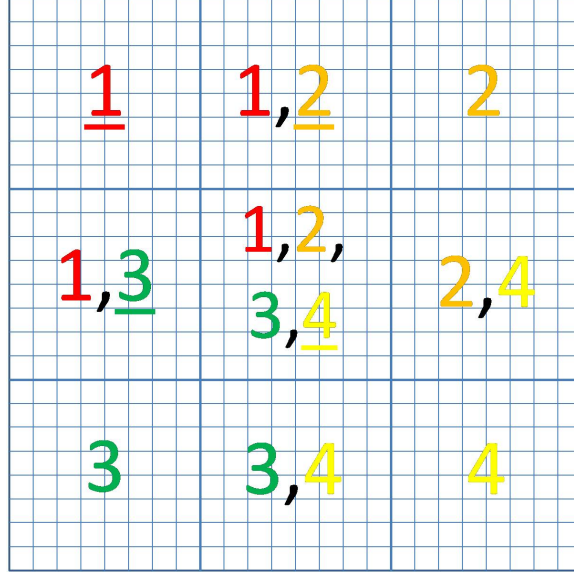


Figure 3.3: **HOG Concatenation Assignment.** The 128-length descriptor is made up of a series of histogram concatenations. First, $4-2 \times 2$ overlapping blocks are formed from each 3×3 patch. Starting with the top-left 2×2 block labeled, 1, the 8-bin histograms from each of the 4 cells are concatenated forming a 32-length vector. Repeating the process for the 2×2 blocks labeled 2–4 and concatenating those vectors in block 1–4 order yields the 128-length descriptor that describes the 3×3 block window.

gradient histograms that have magnitude representation across multiple orientation bins. This is the first threshold method. Next by eliminating those descriptors that don't have peak magnitudes of at least some predetermined percentage greater than the average, the number of features is greatly reduced to a more accurate, not to mention manageable number.

3.1.2 Kernelized Locality-Sensitive Hashing. Kernelized Locality-Sensitive Hashing (KLSH) is similar to standard LSH in that it computes hash functions using random projections; however, all computations are completed in kernel space using only a portion of the data. Each database and query point are transformed to kernel space first; therefore all statistics of the projection functions are unknown and must be estimated. KLSH, thus far has been used to conduct nearest neighbor search on several image databases in both small and large dimension sizes [42].

Just as in LSH, KLSH starts with defining the hash functions. This time, however the database and query collision probabilities are characterized by the amount of similarity between the two points as described by [15].

KLSH Definition *A locality-sensitive hashing scheme is a distribution on a family \mathcal{H} of hash functions operating on a collection of objects, such that for two objects p, q ,*

$$P_r[h(p) = h(q)] = \text{sim}(p, q). \quad (3.3)$$

In other words, the probability of collision between two points is equal to the similarity between them.

$\text{sim}(p, q)$, in this definition is the similarity function and $h(p)$ is a hash function selected at random from \mathcal{H} . The intuition behind this is that when $h(p) = h(q)$, p and q collide in the hash table or are assigned to the same hash. Therefore points that are highly similar have a higher probability of being stored together in the hash table resulting in collision [43]. This definition of LSH is slightly different than the one given previously in the LSH discussion as defined by [31]; however, the concept is the same. This implementation uses the definition as used by [42].

Describing the similarity function in terms of the inner product yields:

$$\text{sim}(p, q) = p^T q \quad (3.4)$$

[15] then expanded on this function defining the locality-sensitive hash function from Equation (3.3) as

$$h_{\vec{r}}(p) = \begin{cases} 1, & \text{if } \vec{r}^T p \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (3.5)$$

where \vec{r} is a random hyperplane from a zero-mean multivariate Gaussian $\mathcal{N}(0, \Sigma_p)$ with the same dimensionality as the input vector p . This implies that each hash

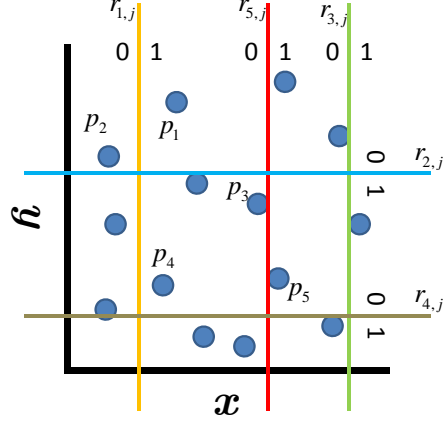


Figure 3.4: **KLSH Projection Example.** The projection of a random vector, \vec{r} on a point, p across dimension, d_t equates to the spatial position of the coordinates in d_t with respect to \vec{r} .

function is built solely from the statistics of the input. The proof showing that this function obeys the locality-sensitive property is detailed in [32]. Since, one hash equates to the sign of one projection on a point p , a series of hashes is formed simply by repeating this process t times. This forms one hash bucket such that,

$$g(p) = \langle h_1(p), \dots, h_t(p), \dots, h_k(p) \rangle. \quad (3.6)$$

To understand the computation of one projection, consider the following example. Fig. 3.4 shows a set of data points, \mathcal{P} in \mathbb{R}^2 space. As in Equation (3.5), \vec{r} is formed from the statistics of p . A random Gaussian multivariate matrix, \mathcal{R} distributed by $\mathcal{N}(0, \Sigma_p)$ is formed. At random, a dimension in p is chosen to project across. $r_1(p)$ as shown projects across dimension 2. Points to the right of this line ($\vec{r}^T p \geq 0$) are assigned a bit of 1, while points to the left ($\vec{r}^T p < 0$) are assigned as 0. Repeating this process t times yields the hashing scheme in column 1 of Table 3.1. All bits pertaining to each individual point are concatenated to form the length, t hash sequence for each bucket. Given $b = 5$, the table shows each hash location in the bucket.

Table 3.1: **KLSH Hashing Scheme.** The hash location for a particular point, p is equal to the concatenated bits in the corresponding column. The hash bucket contains the hashes for each point.

| | p_1 | p_2 | p_3 | p_4 | p_5 |
|-----------|-------|-------|-------|-------|-------|
| $r_{1,j}$ | 1 | 0 | 1 | 1 | 1 |
| $r_{2,j}$ | 0 | 1 | 1 | 1 | 1 |
| $r_{3,j}$ | 1 | 0 | 0 | 0 | 1 |
| $r_{4,j}$ | 0 | 1 | 0 | 1 | 0 |
| $r_{5,j}$ | 1 | 0 | 0 | 0 | 1 |

$$g_j(p) = [h_{1,j_{\vec{r}}}(p_i) \quad \dots \quad h_{k,j_{\vec{r}}}(p_i)] =$$

| | | | | |
|---------|---------|---------|---------|---------|
| [10101] | [01010] | [11000] | [11010] | [11101] |
|---------|---------|---------|---------|---------|

This shows that repeating this process b times yields the following series of hashes for p ,

$$g(p) = \left\{ \begin{array}{cccccc} h_{1,1\vec{r}}(p) & h_{2,1\vec{r}}(p) & \dots & h_{l,1\vec{r}}(p) & \dots & h_{t,1\vec{r}}(p) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{1,j\vec{r}}(p) & h_{2,j\vec{r}}(p) & \dots & h_{l,j\vec{r}}(p) & \dots & h_{t,j\vec{r}}(p) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{1,b\vec{r}}(p) & h_{2,b\vec{r}}(p) & \dots & h_{l,b\vec{r}}(p) & \dots & h_{t,b\vec{r}}(p) \end{array} \right\} \quad (3.7)$$

for $1 < j < b$, $1 < l < t$. In KLSH, all computation is done in kernel space such that the similarity function in Equation (3.4) extends to

$$\text{sim}((x_i, x_j)) = \kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j), \quad (3.8)$$

In this formulation, $\kappa(x_i, x_j)$ is the mapping of points x_i and x_j to kernel space using a predefined kernel function. $\phi(\bullet)$ therefore, is a compilation of the random hyperplane projection hash functions from H using the method just discussed. The problem is that nothing is known about the data while in kernel space to generate \vec{r} from. Therefore, in order to construct the hash function, \vec{r} needs to be constructed such

that $\vec{r}^T \phi(x)$ can be computed directly from the kernel function. Like the standard \vec{r} , it should be approximately Gaussian but allow the function as a whole to be computed using only the kernels. This is accomplished by constructing \vec{r} as a weighted sum of a subset of the database input.

From the central limit theorem discussed in Appendix A.2, consider each kernel data sample, $\phi(x)$ as a vector from some distribution \mathcal{D} with mean, μ and variance, Σ . If t database objects are chosen i.i.d. from \mathcal{D} forming the set S , then a random variable, z_t can be defined as

$$z_t = \frac{1}{k} \sum_{i \in S} \phi(x_i). \quad (3.9)$$

As t grows large, the central limit theorem states that the subsequent random vector \tilde{z}_t , defined by

$$\tilde{z}_t = \sqrt{t}(z_t - \mu) \quad (3.10)$$

has a multivariate Gaussian $\mathcal{N}(0, \Sigma)$. Then the whitening transform is applied to yield

$$\vec{r} = \Sigma^{1/2} \tilde{z}_t \quad (3.11)$$

The hash function then becomes

$$h(\phi(x)) = \begin{cases} 1, & \text{if } \phi(x)^T \Sigma^{1/2} \tilde{z}_t \geq 0 \\ 0, & \text{otherwise} \end{cases}. \quad (3.12)$$

As mentioned previously, since the original data is being represented by the kernel function, statistics such as Σ are unknown and therefore must be estimated via sampling. Selecting n objects from the database enables calculation for the mean and

covariance by eigendecomposition and kernel principal component analysis methods via [62], in which $\Sigma = V\Lambda V^T$. Therefore

$$\Sigma^{1/2} = V\Lambda^{1/2}V^T. \quad (3.13)$$

Define the new hash function as,

$$h(\phi(x)) = \text{sign}(\phi(x)^T V\Lambda^{1/2}V^T \tilde{z}_t) \quad (3.14)$$

This denotes the hash process for a vector of kernel entries. Now, consider the case in which there is a matrix of kernel inputs, K . Its eigendecomposition of $K = U\Theta U^T$ makes this an expanded form of the single kernel input. The non-zero eigenvalues of Λ from Equation (3.13) are the same as the non-zero eigenvalues of Θ . Let v_m be the m -th eigenvector of the covariance matrix and u_m be the m -th eigenvector of the kernel matrix. From [62], compute the projection

$$v_m^T \phi(x) = \sum_{i=1}^n \frac{1}{\sqrt{\theta_m}} u_m(i) \phi(x_i)^T \phi(x). \quad (3.15)$$

in which the $\phi(x_i)$ terms are the n data point samples mention previously. Performing this computation over all m eigenvectors results in

$$h(\phi(x)) = \phi(x)^T V\Lambda^{-1/2}V^T \tilde{z}_t = \sum_{m=1}^n \sqrt{\theta_m} v_m^T \phi(x)^T v_m^T \tilde{z}_t. \quad (3.16)$$

Substituting Equation (3.15) yields

$$\sum_{m=1}^n \sqrt{\theta_m} \left(\sum_{i=1}^n \frac{1}{\sqrt{\theta_m}} u_m(i) \phi(x_i)^T \phi(x) \right) \cdot \left(\sum_{i=1}^n \frac{1}{\sqrt{\theta_m}} u_m(i) \phi(x_i)^T \tilde{z}_t \right). \quad (3.17)$$

Simplifying yields

$$= \sum_{i=1}^n \sum_{j=1}^n (\phi(x_i)^T \phi(x)) \cdot (\phi(x_j)^T \tilde{z}_t) \left(\sum_{m=1}^n \frac{1}{\sqrt{\theta_m}} u_m(i) u_m(j) \right). \quad (3.18)$$

Since $K_{ij}^{-\frac{1}{2}} = \sum_{k=1}^n \frac{1}{\sqrt{\theta_k}} u_k(i) u_k(j)$, further simplification yields

$$h(\phi(x)) = \sum_{i=1}^n w(i) (\phi(x_i)^T \phi(x)), \quad (3.19)$$

where $w(i) = \sum_{j=1}^n K_{ij}^{-1/2} \phi(x_j)^T \tilde{z}_t$.

The Gaussian random vector \vec{r} , becomes $\vec{r} = \sum_{i=1}^n w(i) \phi(x_i)$. This is therefore a weighted sum over the vector inputs from the n database kernel entries. Substituting the random vector of kernels, $\tilde{z}_t = \frac{1}{\sqrt{t}} \sum_{i \in S} \phi(x_i)$ from Equation (3.9) in $w(i)$ yields

$$w(i) = \frac{1}{t} \sum_{j=1}^n \sum_{l \in S} K_{ij}^{-1/2} K_{jl} - \frac{1}{n} \sum_{j=1}^n \sum_{k=1}^n K_{ij}^{-1/2} K_{jk}. \quad (3.20)$$

Since the \sqrt{t} term has no affect on the sign of the hash function, it can be ignored. Final simplification for $w(i)$ is as follows:

1. Define e to be a vector of all ones and e_s to be a vector with ones in the entries corresponding to the indices of S . The resulting $w(i)$ is

$$w = K^{1/2} \left(\frac{1}{k} e_s - \frac{1}{n} e \right). \quad (3.21)$$

2. Therefore, the final hash function for a kernel input is

$$h(\phi(x)) = \text{sign} \left(\sum_{i=1}^n w(i) \kappa(x, x_i) \right) \quad (3.22)$$

in which κ is the mapping of points x and x_i to kernel space. Similar to the hashing discussion at the beginning of this section, each hash will concatenate multiple iterations forming a bucket.

KLSH Algorithm

1. Pre-processing
 - (a) Form kernel matrix, K over n data points from database
 - (b) Form e_s by selecting t indices at random from the set $\{1, \dots, n\}$
 - (c) Each $h_t(\phi(x))$ performs a projection on t^{th} indices
 - (d) Form $w = K^{1/2} \left(\frac{1}{t} e_s - \frac{1}{n} e \right)$
 - (e) Project the vector $w(i)$ onto the point in kernel space in the same manner as the example
 - i. Form hash bucket $g_j(x)$ by assigning bits accordingly
2. Query Process
 - (a) Form the same L hash buckets per Equation (3.23) as done for the database points
 - (b) Use nearest neighbor techniques to answer query

Figure 3.5: **KLSH Algorithm.**

$$g_j(x) = \left[h_1(\phi(x)) \quad h_{2,j}(\phi(x)) \quad \dots \quad h_{t,j}(\phi(x)) \quad \dots \quad h_{k,j}(\phi(x)) \right], (1 < l < t), (1 < j < b) \quad (3.23)$$

To determine the best parameter set, this implementation tests accuracy of query matches throughout multiple iterations, each time varying the number of points in the database, n , the number of hash functions per bucket, b as well as the number of feature points per image, t in each sample to estimate statistics from. The final algorithm detailing this process is in Fig. 3.5.

3.1.3 RANSAC. Given two images of the same scene but translated, the features in each should be the same, however, their locations will be different. In selecting matching features between images, the goal is to eliminate the outliers, or those matches which mathematically appear to be the same according to distance metrics, but visually are not as shown in red in Fig. 3.6.

To handle this task, this implementation uses the RANdom Sampling And Consensus (RANSAC) technique. In the past, RANSAC has been used for tasks such as

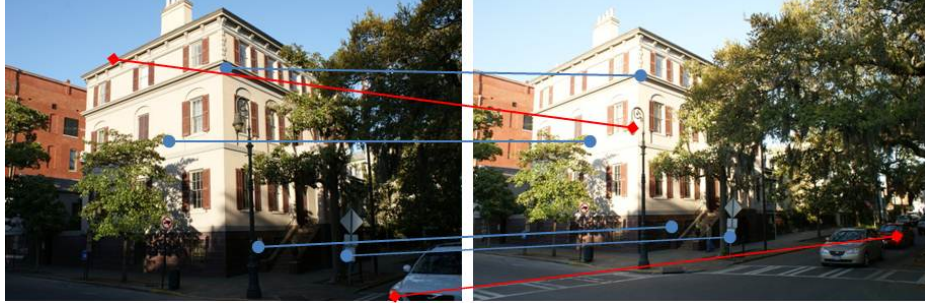


Figure 3.6: Identifying inliers and outliers during feature matching between 2 translated image frames. This is an arbitrary example in which each blue circle or diamond represents any feature type. The blue lines represent correctly matched features, or inliers. The red lines represent feature mismatches or outliers.

model parameter estimation in the presence of outliers [29]. It has recently been used for data association with vision only SLAM using an EKF [17] as well as for agent localization in [63], [30], [59], and [64]. One reason RANSAC is favored over other techniques is its ability to give accurate results even in datasets in which more than 50% of the set are outliers. This percentage is known as the breakdown point. As mentioned previously, the LMS technique falls prey to this limit, as does least squares estimation [80]. To begin, RANSAC is broken into two main steps, hypothesis and test.

In the hypothesis step, a minimal sample set (MSS) of the data is randomly selected. RANSAC differs from other algorithms in that this sample set must be as small as sufficiently possible in order to model the data as opposed to using the entire dataset in least-squares and least median squares.

The test step simply tests how many samples of the entire dataset fit the model. The set of samples that meet this criterion are known as the consensus set (CS), in which larger sets are considered better. These steps are repeated until the probability of finding a better CS falls below a predetermined threshold. Another termination method is to use the best CS out of a specific number of iterations. The disadvantage to this is that in repeating the algorithm until probabilistically complete has the potential risk of no cap on computation time. Furthermore, while the probability

of a good model increases with the number of iterations, there is no guarantee that the optimal solution has been selected when bounding the number of iterations [80]. As this process is random, it can end in different results from one run to the next, however stronger features and many iterations minimize this occurrence.

3.1.3.1 Calculating the Algorithm Termination Point. As just mentioned, the hypothesis and test steps are ran iteratively until the probability of finding a better CS falls below a predetermined threshold or the predetermined number of iterations has been reached. Another method is to combine the two and calculate the number of iterations required produce a given probability threshold. If p_{MSS} is designated as the probability of sampling from the dataset \mathcal{P} a MSS s that produces an accurate estimate of the model parameters. Therefore the probability of producing an MSS with at least one outlier is $1 - p_{MSS}$. Consider the rare situation in which in the T MSSs produced were all contaminated by outliers with probability $(1 - p_{MSS})^T$. Since this probability decreases to 0 as T increases, the solution is to choose the number of iterations, T such that

$$(1 - p_{MSS})^T \leq \epsilon \quad (3.24)$$

where ϵ is a probability threshold. Solving for T and using the ceiling function, $\lceil x \rceil$ yields

$$T = \left\lceil \frac{\log \epsilon}{\log(1 - p_{MSS})} \right\rceil. \quad (3.25)$$

Essentially, T is the smallest number of iterations required to produced the probability term in the ceiling function.

3.1.3.2 Inlier/Outlier Threshold. Once the model is chosen and the points are being compared to determine inliers, the selection is based on a distance threshold. The selection of this threshold is key to the accuracy of the result. Setting

the threshold too high will result in bad models being ranked equally with good ones as shown in Fig. 3.7a. Fig. 3.7b shows that setting as the threshold too low will yield the opposite in that none of the points will fit the model.

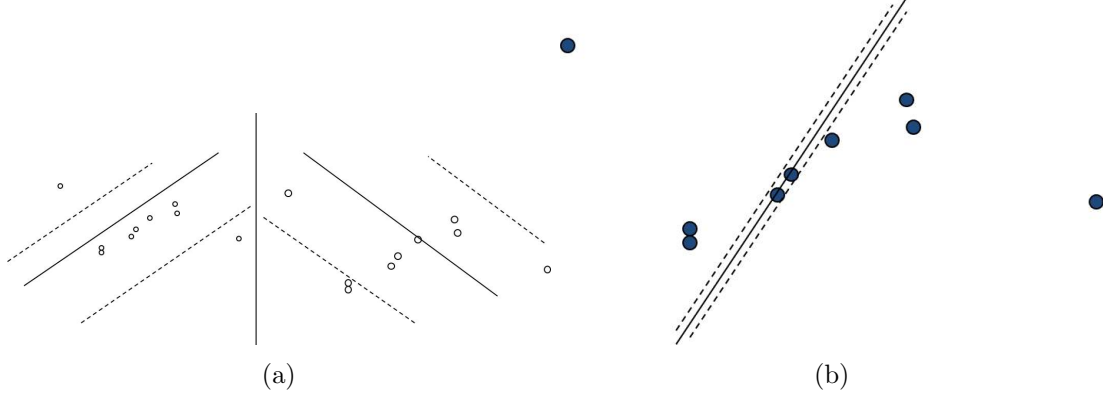


Figure 3.7: **RANSAC distance threshold effects. Example of a threshold set too high (a). Example of a threshold set too low (b) [72].**

3.1.4 Integration into Data Association Implementation. For each image P_0 and P_{trans} , $N \times d$ feature sets are retrieved, in which N is the number of features captured while d is the number of dimensions, 128. First the features in each image are matched comparing all features with each other finding the minimum l_2 Euclidean distance norm. The locations of the feature mappings whose distances are below a threshold are stored and become the dataset as to which RANSAC analyzes. Locations of features from image P_0 are stored in p_0 and locations from P_{trans} in p_{trans} .

As RANSAC is a random process, the MSS is determined by selecting n matching point samples at random with replacement between iterations. In order to represent the correspondence between the two images, a homography must be estimated. See Appendix A.3 for a more details discussion than the derivations that follow. The 2D version is represented by a 3×3 matrix H . This transformation matrix maps the strong features, p'_0 in the first image to the same plane as the corresponding features, p_{trans} in the second image such that

$$\hat{p}_0 = Hp_0 \quad (3.26)$$

in which \hat{p}_0 represents the feature locations of p_0 in the same plane as p_{trans} .

To solve this transformation model, the entries of H are derived by the linear system, $Ah = 0$, which is computed as discussed in Section A.3. A is a $2n \times 9$ matrix in which n is the number of points used for the model. The system is solved by performing singular value decomposition on A where H is derived from the eigenvector, V_i corresponding to the smallest eigenvalue of A for $(1 < i < 9)$.

$$H = \begin{bmatrix} V_{i1} & V_{i2} & V_{i3} \\ V_{i4} & V_{i5} & V_{i6} \\ V_{i7} & V_{i8} & V_{i9} \end{bmatrix} \quad (3.27)$$

This essentially eliminates the translation between the two images allowing for feature to feature comparison among the feature matches using Euclidean distance. Those features that are below a predetermined threshold are considered inliers. This makes up the CS. This process is repeated many times, R noting the number of inliers each iteration. The model that produces the largest consensus set is considered the best choice. The value of R is dependent on how many iterations it takes for the probability of a better inlier result to occur. This probability gets smaller and smaller as the R increases. This implementation evaluates this iteration parameter across many values testing each time for the number of iterations that minimizes the overall count, yet meets a predetermined threshold of accuracy.

3.2 Computing the Data Association

Each input to the KLSH algorithm includes the features of the query image along with the features of all the images being queried according to an orientation heuristic. During preprocessing, the agent orientation at the time of each observation is retrieved from an internal navigation system known as FV-SIFT. This algorithm

separately tracked image features updating a pose estimated from the IMU shown [76]. The agent orientation at each time step an image was taken is notated. Those database image whose orientations match that of the query are input to KLSH. For each query image feature, m image identifiers are returned. The top k identifiers whose feature points in \mathbb{R}^{128} space are closest to the query are output as neighboring matches.

Just as there is uncertainty in the pose at each landmark, there is uncertainty in the data association as well. This can be modeled by the following. For every query feature, there will be at most k -nearest neighbors returned. The accuracy of the output depends on the number of features in each image used for analysis. An increase in the number of features increases query match accuracy but also increases the complexity, memory requirements and runtime. This implementation ran multiple iterations on each dataset, varying in the number of features kept. The following is an example of the nearest neighbor matching process. Fig. 3.8 below shows a sample path through an environment. As shown, each marker represents a landmark at which an image was taken.

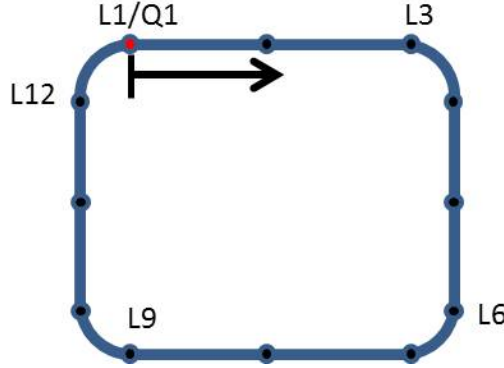


Figure 3.8: **Data Association Computation Example: Sample Path.** $L1 - L12$ are landmarks, or points in the environment at which an image was taken. After completing the first loop and returning to $L1$, query image, $Q1$ is taken, followed by $Q2 - Q12$ assuming the path was repeated.

As the agent repeats the loop and reappears at landmark $L1$, the following data association example begins. At this point, the agent knows that it is at a location that it has seen before, but does not know which landmark that is. This first point

of reappearance is set as query, $Q1$. In order to determine this location relative to landmark position, the image features for query image $Q1$ as well as the features for each landmark $L1 - L12$ are input to the nearest neighbor KLSH algorithm. Tables 3.2-3.5 represent the k -NN search process via KLSH with parameters $k = 4$ and 10 features. Fig. 3.9 shows an illustration of how the features from each image are identified by the image ID (IID). In this arbitrary example, landmarks $L1 - L12$ mentioned earlier, are represented as IIDs 100 – 111 respectively.

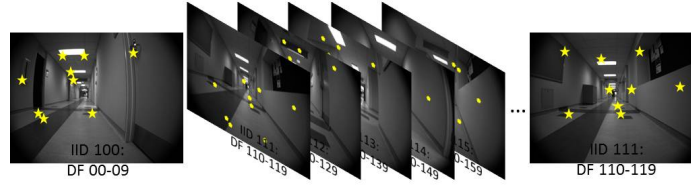


Figure 3.9: **Data Association Computation Example: KLSH input.** This arbitrary example illustrates the database feature identification and association to an image ID. IIDs 100 – 111 correspond with landmarks $L1 - L12$. These along with the query features for a particular query landmark are sent as inputs to the KLSH algorithm. The results for $k = 4$ are shown in Tables 3.2-3.5. Note: The results were arbitrarily chosen to represent process flow and were not actually calculated.

The outputs are the k -nearest neighbors that are most similar to the input query as shown in Table 3.2. Each query feature (QF) matches to $k = 4$ neighboring landmark database features (DF). The image ID for each DF is also notated.

Table 3.2: **Data Association Computation Example: KLSH output.** The outputs are the k -nearest neighbors for each feature.

| | Query Feature # | Matching Feat #/Source Image ID | | | |
|---------------|-----------------|---------------------------------|----------|----------|----------|
| | | NN1 | NN2 | NN3 | NN4 |
| Query Image 1 | QF1 | DF01/100 | DF02/100 | DF24/102 | DF10/101 |
| | QF2 | DF13/101 | DF05/100 | DF59/105 | DF04/100 |
| | QF3 | DF02/100 | DF21/102 | DF31/103 | DF42/104 |
| | QF4 | DF04/100 | DF05/100 | DF09/100 | DF17/101 |
| | QF5 | DF49/104 | DF06/100 | DF36/103 | DF26/102 |
| | QF6 | DF07/100 | DF03/100 | DF08/100 | DF08/100 |
| | QF7 | DF15/101 | DF03/100 | DF54/105 | DF46/104 |
| | QF8 | DF06/100 | DF22/102 | DF04/100 | DF55/105 |
| | QF9 | DF01/100 | DF31/103 | DF29/102 | DF49/104 |
| | QF10 | DF01/100 | DF02/100 | DF51/101 | DF16/101 |

Next in Table 3.3, the number of occurrences of each image ID for each feature match is tallied. These tallies are then normalized with respect to each feature in Table 3.4 to generate a weight to identify the final nearest neighbors.

Table 3.3: **Data Association Computation Example: Identify image representation.**

| | Query Feature # | Source Image IDs | | | | | | | | | | | |
|---------------|-----------------|------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| | | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| Query Image 1 | QF1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF2 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF3 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF4 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF5 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF6 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF7 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF8 | 2 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF9 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF10 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 3.4: **Data Association Computation Example: Normalize image representation across all images.**

| | Query Feature # | Source Image IDs | | | | | | | | | | | |
|---------------|-----------------|------------------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|
| | | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| Query Image 1 | QF1 | 0.5 | 0.25 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF2 | 0.5 | 0.25 | 0 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF3 | 0.25 | 0 | 0.25 | 0.25 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF4 | 0.75 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF5 | 0.25 | 0 | 0.25 | 0.25 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF6 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF7 | 0.25 | 0.25 | 0 | 0 | 0.25 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF8 | 0.5 | 0 | 0.25 | 0 | 0 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF9 | 0.25 | 0 | 0.25 | 0.25 | 0.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | QF10 | 0.5 | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 3.5: **Data Association Computation Example: Query k -NNs. These are the k -NNs to the the query. This is the input to RANSAC.**

| Source Image IDs | 100 | 101 | 102 | 103 |
|------------------|------|-----|------|------|
| Query Image 1 | 4.75 | 1.5 | 1.25 | 0.75 |

Finally, Table 3.5 shows the k -nearest neighboring image IDs that have the most representation as shown by their weightings, across all of the query features. These $k = 4$ images are analyzed further using RANSAC to find an exact match to the query.

3.2.1 An Exact Match From k -NNs. To complete the data association problem, the top k nearest neighbors from the KLSH output, as shown in Table 3.5

are retrieved and analyzed for an exact match using RANSAC. The original query image is compared with each possible match yielding k different inlier results. Recall the number of inliers are those feature matches between images that meet distance thresholds calculable from a good transformation model. After many iterations, the consensus set with the most inliers is selected as the most probabilistic match for the query.

To determine the number of iterations, this algorithm actually followed both methods discussed. An arbitrary number of iterations was made, justified by the accuracy of match. The output is the time stamp of the image that has the most feature matches between the two image spaces.

3.3 Summary

This section has reviewed the background research behind all of the methods used to compute the data association and explained how it is implemented to solve the data association problem in vision-based SLAM. Once features have been extracted through either SIFT or HOG feature detection, hash-based methods via KLSH are used to narrow down the search space in matching a single query feature to an entire database of features. k -nearest neighbor search methods result in each query image being matched with the k -most similar stored database images. The image IDs for each image are then matched with the query side by side using homography-based RANSAC. By correctly modeling the matching features in each image to the system, false matches can be detected, resulting in the identification of the correct image through standard distance techniques. The matches found answer the data association for each instance in an agent's movement throughout the environment. The associations can be extended for use in vision-based SLAM and other image mapping algorithms.

IV. Results & Analysis

This chapter discusses the specific tests conducted to use KLSH to answer the data association when determining agent pose in an environment. First the testing details are discussed including vehicles, test grounds, and parameters used. Next the results of using KLSH are presented and an analysis of its robustness as a viable option to conduct data association is assessed.

4.1 *Testing Procedure*

4.1.1 Terrain. All tests were completed indoors in a hallway environment. The overall path as shown in blue in Fig.4.1 was rectangular in shape with a perimeter of approximately 210m. The hallway had tile flooring and is 2.5m wide. The agent started at the top left blue arrow and traversed the path twice in the counterclockwise direction producing just over 1260 images. From this path two smaller datasets were derived. The path in red represents the traversing of approximately 140m on 3 of the straightaways. 2 legs of this path produced a dataset with 510 images. Finally, the last dataset simulates a path in which the agent made an observation in the form of an image every 5m. The black and yellow circles shown are not to scale but provide a general idea of the 5m distance with respect to the overall floor plan. This 2-lap 5m dataset consists of 135 images.

It should be noted that the hallway used can be classified as office-like. The distinction is made to express volume the traffic, causing inconsistent features in the environment. While the building in which the tests were conducted was not a densely populated area, there were instances of people walking through the hall that provided positive obstacles for the feature algorithm to overcome. In computer vision, a person or object that is apparent in one image and not in the next can have adverse affects on image registration.

4.1.2 Agent. The agent, the Pioneer 2-AT in shown in Fig. 4.3 was driven via remote control. Contained in its setup is all of the hardware required for data

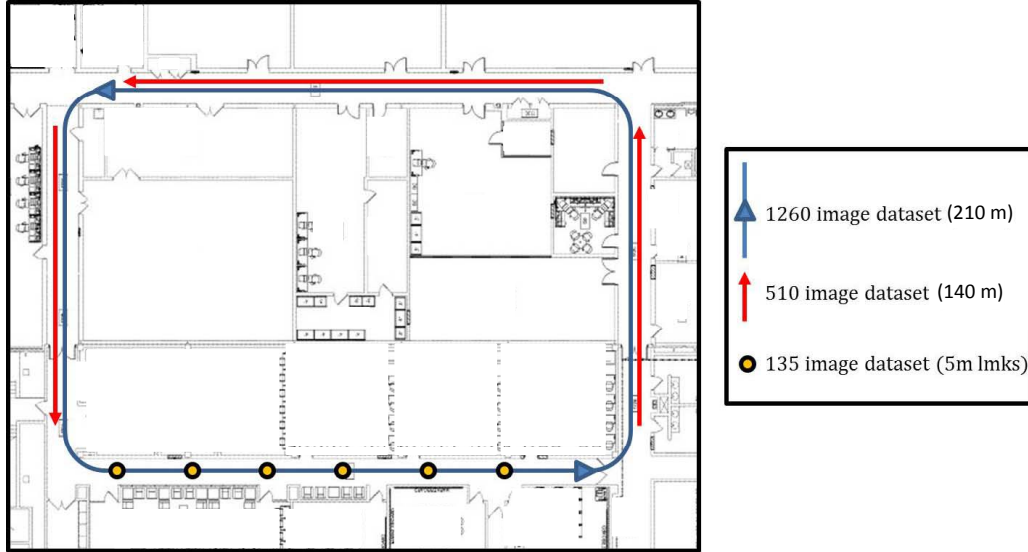


Figure 4.1: Environment Layout.

collection. The hardware used is listed in Fig. 4.2. (Fig. 4.3 does display some hardware not listed. These devices were not used in this implementation.)

Pioneer 2-AT Hardware

- PixeLINK PL-A741 machine vision cameras(2)
- MicroRobotics MIDG II consumer grade microelectricalmechanical systems (MEMS) IMU
- Vision Computer
 - 2.0 GHz Intel Core2Duo processor
 - 4 GB memory
 - Nvidia 9800GTX+ Graphics Processing Unit (GPU)
- Internal Computer
 - Records vehicle odometry

Figure 4.2: **Pioneer 2-AT Hardware.**

The internal computer on board the robot calculates odometry based on the skid-wheel steering of indoor tires with dimensions: 7in diameter, 2in width. The average vehicle speed was about 1 m/s. As the vehicle was driven by remote control, there were occurrences of variance in speed. These were assumed to be negligible in mapping.



Figure 4.3: Pioneer 2-AT vehicle used for data collection. The circled camera on the left of the camera bar was used to conduct data association. The IMU is mounted at the center of the camera bar while the vision computer and communications equipment are mounted towards the rear of the chassis.

4.1.3 Image and Sensor Data Retrieval. The cameras as shown in Fig. 4.3 are placed in stereo mode; however, this implementation only uses images from the slave camera, as denoted by the circle in the figure. It has a 90° field of view and images are taken at a resolution of 1280×960 at a rate of 2 Hz. Taking into account the average speed of 1 m/s, images were captured approximately every 0.5 m. The image retrieval algorithm for this system is actually running a highly parallelized SURF feature extraction program for a FV-SIFT [76]. FV-SIFT is an internal navigation system that tracks image features to update the pose given from inertial measurements [76]. This section of the process is calculated by the GPU which simultaneously allows for the recording of stereo images and IMU data. The vehicle's internal computer records odometry used for pose estimation and links to the external computer via Ethernet cable for time synchronization and communication with FV-SIFT.

For the purposes of this implementation, the images from the slave camera as well as the orientation information from FV-SIFT are used. The images are saved as

portable graymap files (.pgm) in Netpbm format, while the other data collected/calculated are written as logs to text files.

4.2 *Implementation Details & Parameters*

The images are analyzed for features using a compiled executable for SIFT features [46] and Matlab[®] 2010a for HOG. All feature descriptors and parameter information are stored as Matlab[®] .mat files. As this implementation does not include a loop closure heuristic, the image data is manually split between database and query by visually determining the identifiers of the first and second loop. Each query is then processed by the Matlab[®] written Data Association algorithm. All processing was completed on dual core processors with 4 GB of memory.

4.2.1 Memory Obstacles. As mentioned previously, this implementation was run on a dual-core computer with limited memory. For this as well as possible future use in online mobile algorithms, it is important to add heuristics that constrain the number of features in the overall database or those sent to the KLSH algorithm at one time.

4.2.1.1 Database Reduction Heuristic . The feature database as a whole can have hundreds of thousands of features that must be sifted through in order to find a match to a given query. By dividing the feature database into classes, the number of features to be hashed for a particular query analysis is reduced. One such class is based on heading. The features going in one direction will be different than the features that are seen going the opposite direction in the same environment. Therefore, a query can only be matched to features whose difference in orientation is within tolerance. This implementation assigns the heading categories shown in Table 4.1 to each feature based on heading information gathered from the FV-SIFT output. This results in each query being matched with features in 1 of 4 KLSH-driven hash tables.

Table 4.1: **Database Reduction Heuristic: Orientation Classification. This is the orientation classification used to classify the features in each image.**

| Heading Range | Category |
|---------------|----------|
| 0 – 51 | 1 |
| 52 – 143 | 2 |
| 144 – 225 | 3 |
| 226 – 321 | 4 |
| 322 – 359 | 1 |

4.2.1.2 Multiple KLSH runs per kernel. While trimming down the database size with orientation heuristics helps, this implementation still ran into memory issues. This implementation simply broke the KLSH hashing process into multiple iterations. Define $N_{F,\theta}$ as the number of database features corresponding to the orientation of q . Dividing $N_{F,\theta}$ throughout ι iterations produces

$$N'_{F,\theta} = \frac{N_{F,\theta}}{\iota} \quad (4.1)$$

features per iteration. This reduces the complexity required in hashing $N_{F,\theta}$ points to b tables to ι iterations of $N'_{F,\theta}$ points to b tables. This has a direct affect on the kernel matrix size. Consider $N_{F,\theta} = 10K, \iota = 20$. By

$$\frac{N_{F,\theta}'}{\iota} = \frac{10K}{\iota} = 500, \quad (4.2)$$

the kernel matrix decreases in size from $10K \times 10K$ to $20 \cdot 500 \times 500$ kernel matrices, thereby lessening memory requirements to that of a standard computer when running implementation. Furthermore, the speed at which these table hashings are computed is increased.

4.2.2 Kernel Matrix Generation. [42] reported that both the Gaussian radial basis function (RBF) and Chi-Squared kernels were used in testing on multiple datasets. Therefore this implementation tested on both as well.

Gaussian RBF kernel

$$\kappa(p, q) = e^{-\frac{\|p-q\|^2}{2\sigma^2}}. \quad (4.3)$$

Chi-Squared kernel

$$\kappa(p, q) = 1 - \sum_{i=1}^n \frac{(p_i - q_i)^2}{\frac{1}{2}(p_i + q_i)}. \quad (4.4)$$

In each case, p and q are the $N \times d$ matrices of database and query features in which N and d are the number of features and number of dimensions, respectively. For a given KLSH iteration, the query inputs are the features from the query image, $N_q \times d$ as well as the comparison database features, $N_p \times d$. Naturally, $N_p \gg N_q$. In order to compute κ , these matrices must be the same size. The solution to this is to build the q kernel input such that it matches the dimensions of p by,

$$\begin{bmatrix} N_q \times d \text{ matrix of query features} \\ (N_p - N_q) \times d \text{ matrix of ones} \end{bmatrix} \quad (4.5)$$

4.3 Results

To determine the best fit for parameters and test overall performance of the KLSH algorithm, the following tests were run:

1. Parameter sweep: Best choice for KLSH hash parameters n , b , and t
2. Parameter sweep: best choice of RANSAC iterations
3. Feature extraction comparison (SIFT vs HOG)
4. Parameter sweep: best choice of k for NN search
5. Data association performance: required trade-offs between memory use, accuracy, and speed

In using parameter sweeps to determine the best set, multiple tests were accomplished varying the parameters such that an optimal solution could be calculated or extracted

from the results. The best choice for each sweep is then used in determining the overall performance of the algorithm in Section 4.3.5. This performance as well as the underlying determination of a good parameter is based on an accuracy of match metric. Once KLSH and RANSAC have run and an answer is provided for a given query, the accuracy of that match is calculated based on a truth table. During preprocessing, the first and second loop images were aligned by time stamp. A truth table among the two categories of time stamps was generated, then verified visually. Recall the image capture rate of 2 Hz. For the large full loop dataset, the assumption was made that if the vehicle moved at an average speed of 1 m/s then the avg distance between 2 images is 0.5 m. The desired position error is ± 2 m. This therefore equates to the requirement of a matching image time stamp to be within ± 2 seconds from the truth to be considered a positive match.

4.3.1 KLSH parameters. Determining a good parameter set is key in deriving the neighboring features in the KLSH output. Such a parameter set optimizes accuracy of match with speed, minimal memory usage, and complexity. To start, the value of n , or the number of features in the database to be sampled, was analyzed. In contrast from [42] and leaving this parameter constant at 300 for all database sizes, this implementation left the determination of the size of n up to the algorithm heuristics mentioned in Section 4.2.1.1. In other words, n varies based on the number of features, $N_{F,\theta}$ in the database. Since the KLSH algorithm is split into ι iterations of b hash tables, the size of n changes with every iteration. Furthermore, in an actual SLAM implementation, the size of the entire database, N will require some constraints, the overall size will continue to grow the longer the agent travels in the environment.

To analyze the effect the parameters have on an individual match solution, this implementation conducted a parameter sweep varying the number of hash buckets or tables, b , the number of hashes per table, t as well as the size of the dataset, n . The test was then conducted as follows:

1. Manually select a query and a matching database image
2. Insert these as well as the features of N_I neighboring images into KLSH, $N_I \in \{7, 10, 20, 30, 40, 50\}$
 - (a) 50 features per image are used.
 - (b) The first iteration uses parameter $N_I = 7$ producing 350 features
 - (c) The process is repeated for a total of 6 iterations, each increasing the number of images added to the database, thereby varying n by $350 \leq n \leq 2500$ with N_I
3. Run KLSH 2500 times per iteration of N varying b, t by $1 < b < 500, 1 < t < 300$
4. Assign a 1 if the correct match was one of 7 NNs and a 0 otherwise
 - (a) A correct match is an image whose IID is within 2 secs of the exact match
5. These steps were conducted using SIFT features and a KLSH Gaussian kernel retrieving 7 NNs

Fig. 4.4 shows the KLSH performance as a function of the parameters n , t , and b . Each plot represents an increasing sample size, n , while the rows and columns represent the increasing number of hash tables and hash bits per table respectively. For a given b, t a white color assignment (1) means that there was a correct match to the query within the top 7 nearest neighbors returned. A black color assignment (0) is indicative of a false match.

These results show that when the dataset size is small, the probability of match despite the hash parameters, b and t is large. As either b or t increases, KLSH performance increases; however an increase in both parameters simultaneously, degrades performance trends severely. [42] found $t = 30, b = 300$ to be a successful parameter set based on performance in the low percentage of database features required in searching for a query. This implementation used the above plots to measure these parameters as well. In 5 of the 6 iterations shown, $t = 30, b = 300$ produced a correct match. These parameters also optimize memory efficiency as well. It was mentioned that either b

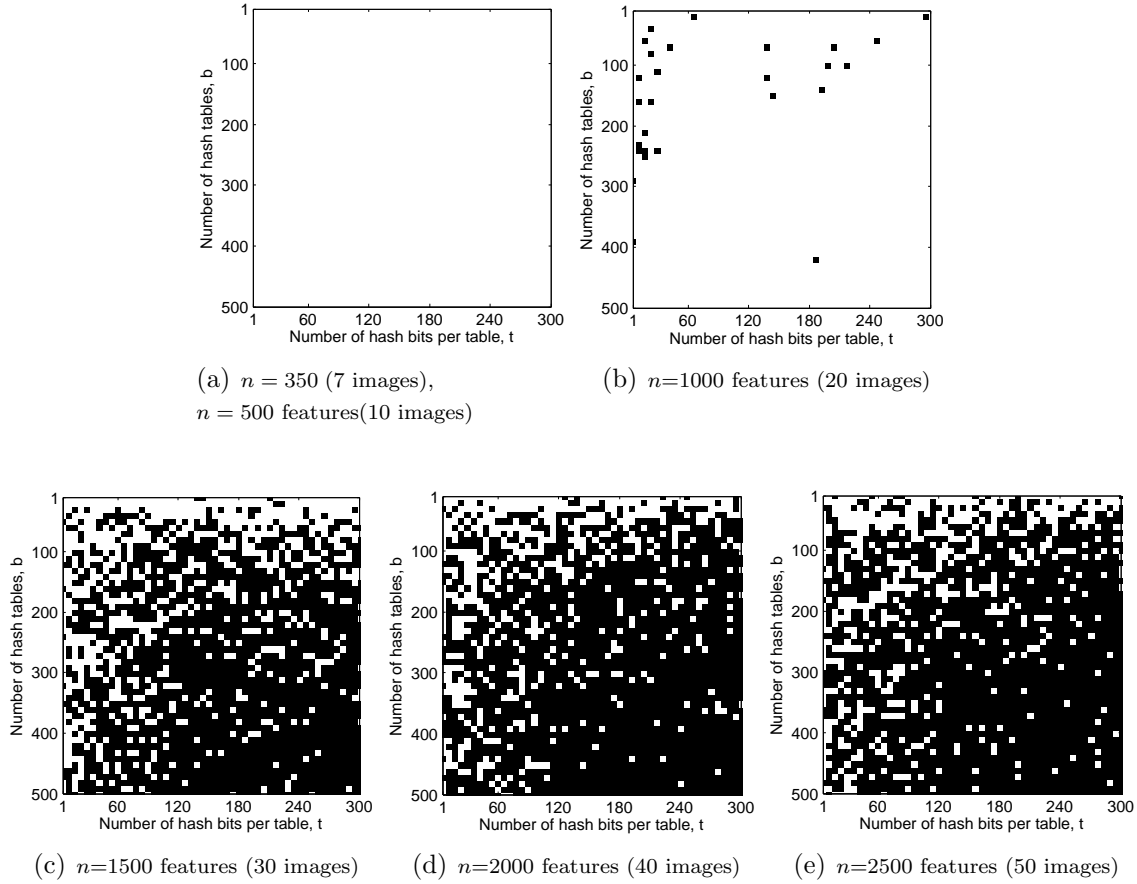


Figure 4.4: **KLSH performance varying n , sample sizes, t , hash bits per table and b , hash tables.** A white color assignment means there was a correct KLSH query match among the 7 NNs returned, while a black assignment means there was no match.

and t should be inversely proportional in size to produce sufficient accuracy. Since the t term increases exponentially in the memory requirement of $O(N(t^2b))$, the set of $t = 30, b = 300$ is much more efficient than $b \ll t$. The plots also show that at low values of b and t perform well as well. For instance, $b = t = 10$ not only produces good accuracy throughout all database sizes, but also optimizes the memory required. The parameter set used for testing was in accordance with [42] at $b = 300, t = 30$ and $n = f(N_F)$. In conclusion to the sweep conducted, the optimal parameter set is set at $b = t = 10$.

4.3.2 RANSAC Parameter, R . The number of iterations, R , in a typical RANSAC algorithm is a function of the probability there is a better consensus set than discovered thus far. The greater the number of iterations is directly proportional to the high probability that the best set has been reviewed. Recall the calculation for the minimum number of iterations required in Equation (3.25). Instead of calculating this parameter directly, a parameter sweep was conducted varying the number of iterations and calculating the accuracy of match. Fig. 4.5 shows the data association accuracy using RANSAC varying the number of iterations between 10 – 500. For this test, 100 SIFT features were used from a 1260 image dataset and 7 KLSH nearest neighbors hashed with parameters $b = 300, t = 30$ from a Gaussian kernel were input to RANSAC with the features from each query.

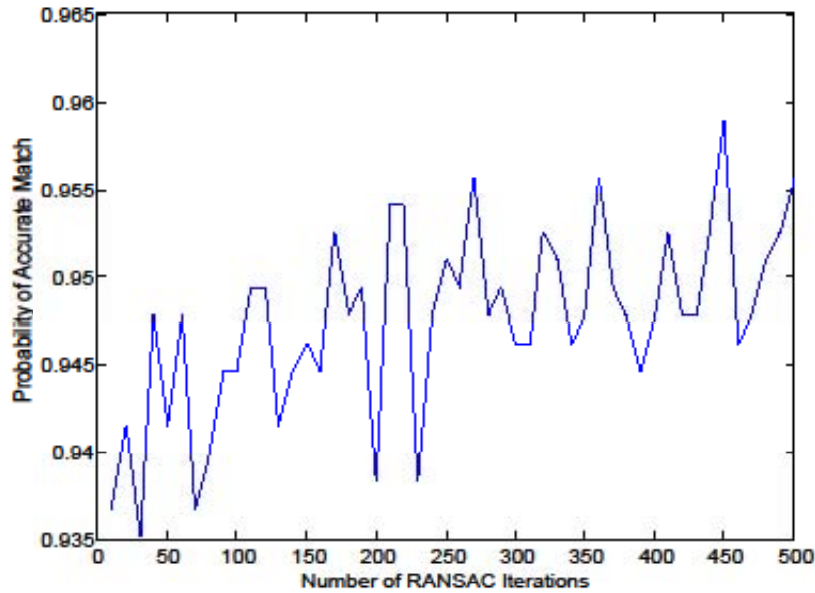
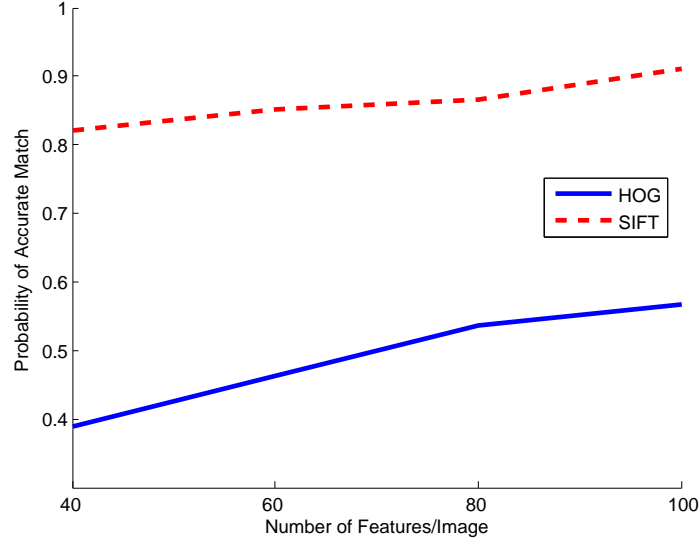


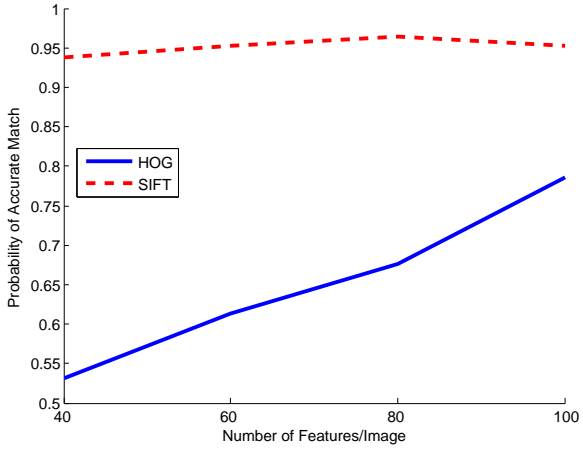
Figure 4.5: **RANSAC iteration parameter sweep.** This sweep shows the accuracy of match in varying the number of RANSAC iterations. This test was conducted using the top 100 SIFT features and 7 KLSH NNs from a dataset of a combined 1260 query and database images.

As shown in Fig. 4.5, for any number of iterations above 225, the probability of accuracy averages at 0.95. This is a commonly used algorithm termination critereon [71], [23]. Based on this result, the number of RANSAC iterations was set at $R=275$. This value sufficiently optimizes algorithm run time as well as accuacy.

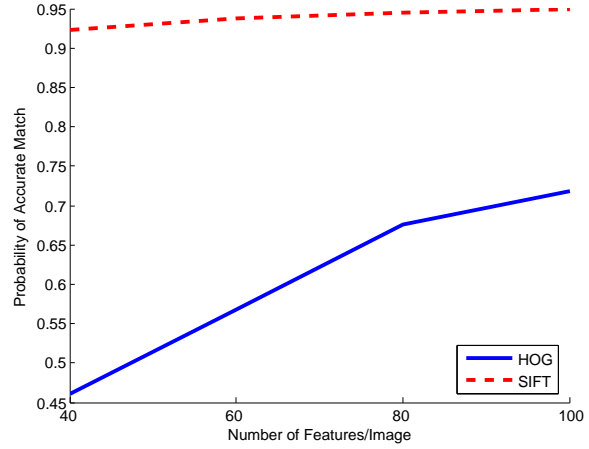
4.3.3 SIFT vs HOG Features. The HOG descriptor vector does not perform as well on larger sized images as it has previously on lower resolution images. The accuracy of match for the HOG descriptor while still suitable, does not compare with the accuracy of SIFT. Fig. 4.6 below shows the accuracy of match from each dataset while varying the number of HOG (solid) and SIFT (dashed) features per image.



(a) 135 image dataset



(b) 510 image dataset



(c) 1260 image dataset

Figure 4.6: **SIFT vs HOG.** These plots show the accuracy of match while varying the number of HOG and SIFT features used in KLSH. These results were produced from a Gaussian kernel, using 7 NNs.

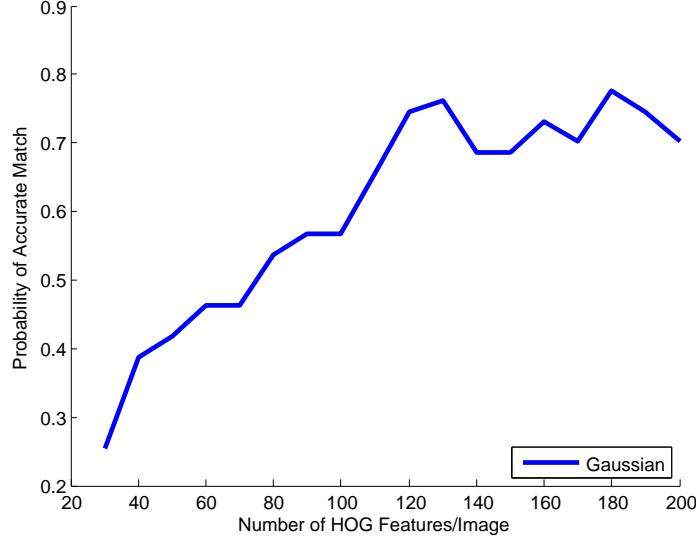


Figure 4.7: **Implementation Performance: Probability of Accurate Match with Varying Number of HOG Features.** This plot shows that the accuracy of match increases with the number of HOG features used per image. This test was completed using HOG features with a Gaussian kernel on the 135 image dataset.

The results shown were run using a Gaussian kernel taking the top 7 KLSH nearest neighbors hashed with parameters $b = 300$ and $t = 30$. In each dataset, the HOG accuracy of match increases with the number of features. This can further be seen in Fig. 4.7. Therefore, in order to obtain accuracy comparable to that of SIFT, more features are required. Continuously increasing led to HOG probabilities of accuracy of 0.90+ with 250+ features. The requirement of twice as many SIFT features essentially doubles the memory requirements, $O(2N_S(t^2b))$, where N_S is the number of SIFT features.

4.3.4 Best Choice of NN. To find the best choice for the number of nearest neighbors to retrieve from KLSH, a parameters sweep was conducted analyzing performance of 1 – 20 nearest neighbors. Fig. 4.8 shows the accuracy at each parameter throughout all datasets.

For all datasets, there is an average peak of accuracy around 5 nearest neighbors. This shows that no matter what the size of the dataset, this is a good choice for k .

High accuracy was also achieved at $k = 1$ nearest neighbor. The difference in this accuracy for $k = 1$ and 5 can in most cases be considered negligible depending on the application. Final determination on this parameter essentially comes down to the compromises that can be accepted between speed and memory usage vs accuracy. This means that given an entire dataset, the search for an exact match to a query can be reduced to searching 5 images rather than all. The resulting complexity therefore is a trade-off of an increase from $O(N_F(t^2b))$ for $k = 1$ to $O(5N_F(t^2b))$ for $k = 5$

4.3.5 Data Association Performance. Next the parameters established in the previous sections were used to conduct the data association on each dataset varying the number of features. Fig. 4.9 shows plots of the number of features vs accuracy for both the Gaussian and Chi-Squared kernels for all datasets.

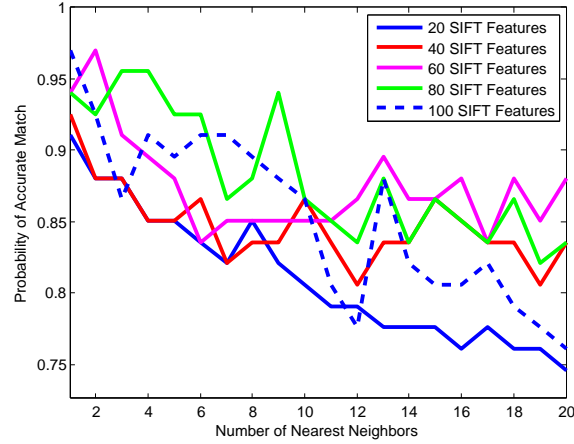
In all datasets, the Gaussian kernel surpassed the performance of that of Chi-Squared. While the accuracy in all cases increases with the increase in the number of features, KLSH via Gaussian kernel performs at 0.95 probability of accuracy with 60 features and above in larger datasets.

4.3.6 Flaws in Accuracy. The plots in Fig. 4.9 show that as the datasets and number of features both increase, the accuracy does as well. In looking specifically at each association made, it was noticed that, in general, it was the same positions in the environment that were being miss-associated. Therefore the degradation in performance is more environment specific than capability of the algorithm. There are always going to be certain areas in environments that features will have a hard time picking up. The key is to optimize feature extraction methods to minimize these occurrences.

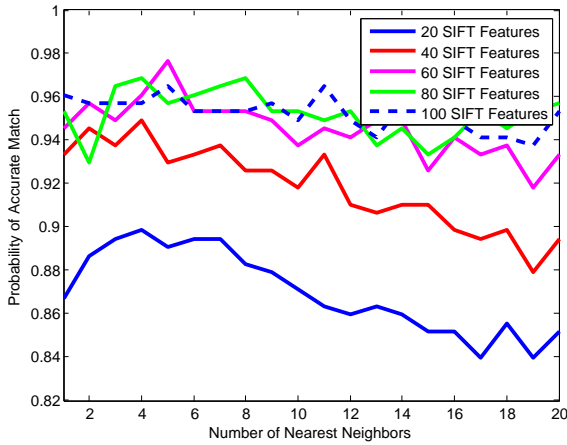
4.4 Summary

This chapter has reviewed test setup and completion. The KLSH hash parameters used produced results answering the data association with metric level precision.

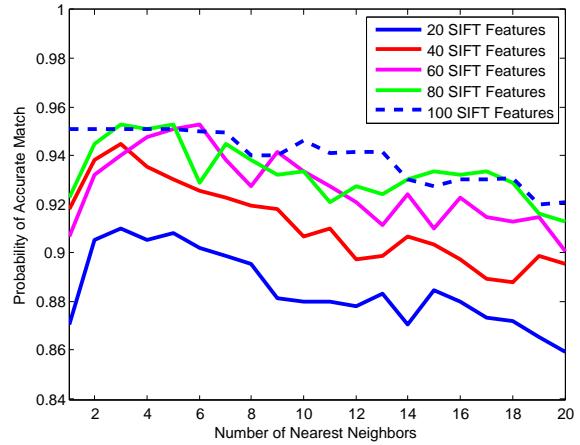
The SIFT feature detector outperformed HOG in accuracy by requiring less features per image to hash similar points. This difference in feature requirement is double the amount of memory required to produce hash tables than needed for SIFT. The Gaussian kernel was found to perform better than the Chi-Squared kernel throughout all dataset sizes. Like the HOG features, however, the Chi-Squared kernel does improve in accuracy as the number of features grows. This in turn increases the complexity, and memory requirements. Finally this implementation found that the number of nearest neighbors required to achieve optimum accuracy can be narrowed between $k = 1$ and 5. Although the accuracy is higher at $k = 5$, there is more of a requirement for memory storage, while the opposite is true for $k = 1$. Here, speed and memory usage is optimized with a slight trade-off in accuracy. The determination for which situation is more ideal is problem specific.



(a) 135 image dataset

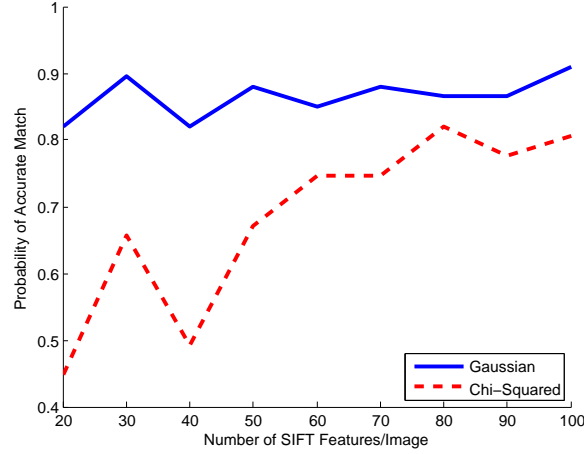


(b) 510 image dataset

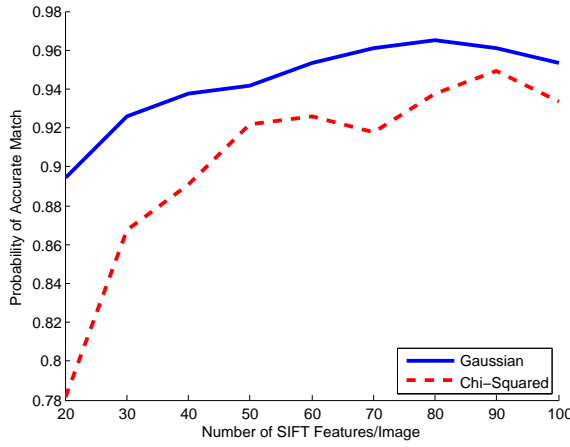


(c) 1260 image dataset

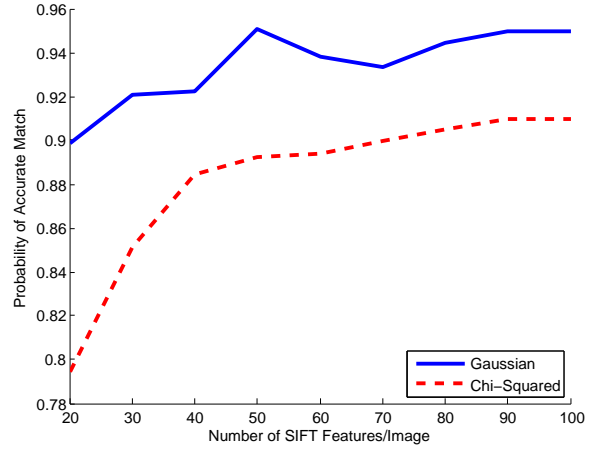
Figure 4.8: **Parameter Sweep: KLSH k .** This plot shows the accuracy of match produced with a variation in the number of nearest neighbors kept. This test was completed using different numbers of SIFT features, according to the legend, with a Gaussian kernel.



(a) 135 image dataset



(b) 510 image dataset



(c) 1260 image dataset

Figure 4.9: **Implementation Performance: Gaussian vs Chi-Squared kernel comparison.** These plots show the accuracy of match while varying the number of SIFT features used in KLSH for both the Gaussian (blue) and Chi-Squared (red) kernels.

V. Conclusion & Future Work

This research has presented an implementation of the hash table-based nearest neighbor search method, KLSH to conduct the data association. The performance has shown to answer the data association of a pose with metric level accuracy within a 2 m. This method therefore can be implemented as the data association solution in a vision-SLAM algorithm. This section discusses the pros and cons to this method as well as the areas for future work found to be required.

5.1 Conclusions

5.1.1 KLSH Performance. KLSH has shown to be a strong nearest neighbor algorithm resulting in optimal accuracy narrowing down the search space needed to answer the data association. The Chi-Squared kernel used in conjunction with SIFT features in [42] did not perform as well in this implementation. The Gaussian kernel performed better by far. Since images were taken at a rate of approximately 2 Hz with an approximate velocity of 1 m/s, a correct data association provides metric level precision down to 2 m. This is a remarkable capability that can be implemented to produce accurate metric SLAM maps.

5.1.2 SIFT vs HOG. This research tests the HOG feature detector’s robustness in using for image navigation purposes. In general, its use in facial and object recognition has been found to produce better results than in this hashed based data association implementation. The accuracy of match for a particular query was found to be lower than that of SIFT but increased with the increase in features per image. The computational complexity and memory requirements for this increase in features in most problems will not warrant its use. When online capability is the ultimate goal, maximum speed and minimum memory usage trade-offs must be optimized. The SIFT feature detector produced matches with greater than 0.90 probability of accuracy on larger datasets with as few as 20 features per image. This yields a memory usage of $O(20N_{F,\theta}(t^2b))$.

5.1.3 Evaluation. Aside from the fact that this implementation was done in **Matlab**[®], the overall process is, however time consuming. By adding database size heuristics and dividing hash table calculations into multiple iterations, the speed of the system was greatly increased without loss in efficiency thereby optimizing use in image mapping. The goal of online capability may be able to be accomplished given the number of features used per image remains relatively low. System constraints are needed, however to determine an actual requirement. Finally, it was determined that there is a trade-off in terms of optimization in the number of nearest neighbors required. As the number of nearest neighbors increased past 5, the accuracy decreased. Therefore, at most, KLSH was found to require no more than 5 images to search in matching a query. In the case of the 1260 image dataset with over 630 database images, that's only 1% of the entire database. It was also discovered however, that at $k=1$, while the probability of accuracy was not as high it was still good, above 0.90 in most cases. This tradeoff in accuracy saves computation time and memory by searching through just 1 image instead of all 5. The conditions warranting the tradeoff are problem dependent but in most cases the difference in accuracy may be negligible. The downfall to KLSH is that as the database size continues to grow, speed will decrease and memory usage will increase. While it was shown that the number of hash tables and bit length of each need not change with the increase in database size, memory usage increases by t^2b with each added feature.

5.2 Future Work

This implementation unfortunately wasn't able to compare the k -d tree method using the same datasets tested on. Therefore an accurate comparison of which method is better could not be accomplished. This is the first area that needs to be addressed in future work. This implementation also needs to be tested in a vision-SLAM implementation and optimized for use online. While the dimensionality required to accurately describe image features has always been much larger than that of other sensor methods, this implementation performs well optimizing accuracy across all

database sizes while searching a fraction of a percentage of the data. The resulting map produced in SLAM will determine whether or not the degradations in accuracy found would impede the overall localization determination. Areas of concern for use online lie in database size. Further heuristics are required to limit the database size while still being able to accurately associate similar points. Furthermore, loop closure techniques need to be implemented when completing in conjunction with SLAM.

While this algorithm performs well at a low number of features, the system could be further optimized with the use of a region or interest point detector that uses general position estimation techniques to estimate a feature's position in the next frame. Tracking fewer features from image to image like this as is done in SLAM will lower the overall number of features in the database. This will not speed the algorithm in terms of dimensionality complexity but will aid in lowering the number entries in the overall database. Another method to decrease the size of the database is to use variations of features as in [18]. Similar to SLAM feature tracking techniques, when a feature is tracked in an environment, variations of that feature in subsequent scenes are combined and associated with a covariance. This ensures that changes in changes in feature appearance as it gets closer don't hinder tracking and data association. This also minimizes the number of additional features that need to be saved in the database when tracking features through the environment.

Appendix A. Supplementary Math

A.1 Hamming Distance Metric

Given a dataset P , let C be the largest coordinate across all dimensions and all samples. This allows \mathcal{P} to be embedded into the Hamming cube with $d' = Cd$. This is done by representing each point p by a binary vector in the form of the unary representation. The unary representation of a number, x_1 is a vector of ones with $\text{length}(x_1)$ followed by $C - x_1$ zeros.

If $C = 10, x_1 = 5$, the $\text{Unary}_C(x_1) = [1111100000]$

The resulting binary representation of a point in P is

$$v(p) = \text{Unary}_C(x_1), \dots, \text{Unary}_C(x_d). \quad (\text{A.1})$$

Consider the 2D, point $p = [3 \ 5]$. The resulting Unary representation based on $C = 5$ is:

$$v(p) = [1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1] \quad (\text{A.2})$$

The Hamming distance, d_H of points p, q in the set $\{1, \dots, d'\}$ is

$$D(p, q) = d_H(v(p), v(p)) \quad (\text{A.3})$$

in which the function d_H is the sum of the XOR of the input terms.

A.2 Central Limit Theorem

Consider a sequence of i.i.d. random variables each with a finite mean, μ and a finite non-zero variance, σ^2 . The sum of the first n is represented by

$$S_n = X_1 + X_2 \dots + X_n \quad (\text{A.4})$$

A zero-mean, unit variance random variable, Z_n can then be calculated as

$$Z_n = \frac{(S_n - n\mu)}{\sigma\sqrt{n}} \quad (\text{A.5})$$

The central limit theorem states that as the number of states, n in a sample becomes large, the cdf of the normalized S_n will resemble that of a Gaussian random variable.

$$\lim_{n \rightarrow \infty} P[Z_n \leq z] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z e^{-\frac{x^2}{2}} dx \quad (\text{A.6})$$

This Gaussian approximation holds true for any distribution as long as there is a finite mean, finite non-zero variance and a properly normalized sum [45].

A.3 2D Homography Estimation



Figure A.1: **Homography Example.**

Consider the 2 images in Fig. A.1, P and P' taken of the same scene who's only differences may include rotation, translation and scale. Since the points in these two images are in different planes or spaces, linear comparisons based on position cannot be completed. A homography is a point to point mapping between two spaces. In this example, a homography enables this pair of images to be compared to complete tasks such as seeing how alike they are or more commonly, image mosaicking. This enables multiple images of the same scene to be "pieced together" in the event they were cropped apart as in Fig. A.2 or paired together to produce wide angle panoramic views. This type of image registration is known as image mosaicking.

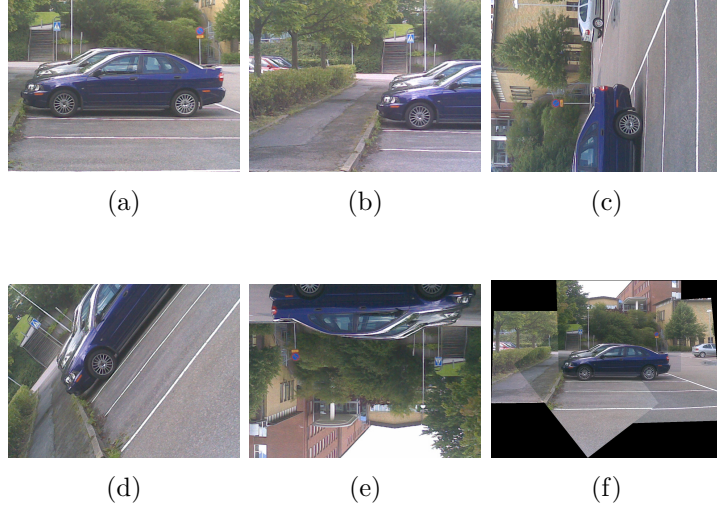


Figure A.2: **Mosaicking example. Original image (a). Various crops with rotation (b-e). Mosaicked image (f)**

Fig. A.2 is an example of the image scenario discussed above. The two images are from the same scene however, the second varies in translation and scale. The following derivations calculate the homography estimation for mapping two images [23].

The mapping of the points p in the first image to the corresponding points p' in the second image defines the homography as

$$wp' = Hp \quad (\text{A.7})$$

where H is a 3×3 transformation matrix. It is important to note that the scale between the images is arbitrary and not defined in the matrix leaving 8 degrees of freedom. A 2D point has 2 degrees of freedom per point mapping. Therefore a minimum of 4 points are required to solve. The system can be solved with more than 4 points; this is known as an overdetermined system in which an approximate solution can be determined by least squares.

Let $p = (x, y, 1)$ and $p' = (x', y', 1)$ be corresponding points in images P and P' . Solving the homography estimation given in equation (A.7) yields

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (\text{A.8})$$

As mention above, the scale parameter cancels when writing out the equations leaving

$$\begin{aligned} h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - x' &= 0 \\ h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' - y' &= 0 \end{aligned} \quad (\text{A.9})$$

Expressing this as the linear system $Ah = 0$ yields the following $2n \times 9$ matrix A , in which n is the number of points used to calculate the homography in each image

$$A = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 & -y'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx'_n & -y_nx'_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_ny'_n & -y_ny'_n & -y'_n \end{pmatrix}. \quad (\text{A.10})$$

h , then is a 9×1 vector comprising the terms of the homography matrix H . Solving for H is now accomplished using singular value decomposition on A yielding

$$A = U\Sigma V' \quad (\text{A.11})$$

in which U and V are the eigenvectors of AA' and $A'A$ respectively and Σ , the singular values which are the square roots of the above eigenvectors. The h vector is equal to the eigenvector corresponding to the smallest eigenvalue of A . This value is 0

if the system is exactly determined, or has only 4 points, or is closest to 0 in an overdetermined systems.

The h vector forms the homography matrix, H as follows:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}. \quad (\text{A.12})$$

Bibliography

1. Andoni, A. “E2LSH: Exact Euclidean Locality-Sensitive Hashing”. Implementation available at, 2004. URL <http://web.mit.edu/andoni/www/LSH/index.html>.
2. Andoni, A., M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. *Locality-Sensitive Hashing Using Stable Distributions*, Chapter 3, 61–72. MIT Press, 2006.
3. Andoni, A. and P. Indyk. “Near-Optimal Hashing Algorithms for Near Neighbor Approximate Nearest Neighbor in High Dimensions”. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, 459–468. Berkeley, CA, Oct 2006.
4. Andriluka, M., S. Koklbrecher, J. Meyer, K. Petersen, P. Schnitzspan, and O. V. Stryk. *RoboCupRescue 2010 - Robot League Team*. Technical report, Technische Universitt Darmstadt, 2010.
5. Barfoot, T. “Online Visual Motion Estimation Using FastSLAM with SIFT Features”. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 579–585. 2005.
6. Bay, H., A. Ess, T. Tuytelaars, and L. V. Gool. “SURF: Speeded Up Robust Features”. *Journal of Computer Vision and Image Understanding*, 110(3):346–359, 2008.
7. Beis, J. S. and D. G. Lowe. “Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces”. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1000–1006. San Juan, 1997.
8. Beis, J. S. and D. G. Lowe. “Indexing Without Invariants in 3D Object Recognition”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 21, 1000–1015. Oct 1999.
9. Bennewit, M., C. Stachniss, W. Burgard, and S. Behnke. “Metric Localization with Scale-Invariant Visual Features Using a Single Perspective Camera”. In *Proceedings of the European Robotics Symposium on Springer Tracts in Advanced Robotics*, Volume 22, 143–157. 2006.
10. Biggs, K. *Real-Time Mapping Using Stereoscopic Vision Optimization*. Master’s Thesis, Wright-Patterson: Air Force Institute of Technology, 2005.
11. Booiij, O., Z. Zivkovic, and B. Krose. “Sampling in Image Space for Vision-Based Slam”. In *Proceedings of the Workshop on Robotics: Science and Systems*, 1–8. Zurich, Switzerland, Jun 2008.
12. Brooks, A. *Improved Multispectral Skin Detection and its Application to Search Space Reduction for Histograms of Oriented Gradients-based Dismount Detection*. Master’s Thesis, Wright-Patterson AFB: Air Force Institute of Technology, 2010.

13. Cai, Z., J. Cao, L. Sun, and M. Li. “Novel RBPF for Mobile Robot SLAM Using Stereo Vision”. *Journal on Artificial Intelligence*, 1(1):1–11, 2008.
14. Carneiro, G. and A. D. Jepson. “Multi-scale Phase-based Local Features”. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Volume 1, I-736 – I-743. 2003.
15. Charikar, M. “Similarity Estimation Techniques From Rounding Algorithms”. *Proceedings of the 34th Annual Conference on ACM Symposium on Theory of Computing*, 380–388. 2002.
16. Chen, Z., J. Samarabandu, and R. Rodrigo. “Recent Advances in Simultaneous Localization and Mapbuilding Using Computer Vision”. *Journal on Advanced Robotics*, 21(3-4):233–265, 2007.
17. Civera, J., O. Grasa, A. Davison, and J. M. M. Montiel. “1-Point RANSAC for EKF-Based Structure from Motion”. In *Proceedings of the International Conference on Intelligent Robots and Systems*. 2009.
18. Clipp, B., J. Lim, JM. Frahm, and M. Pollefeys. “Parallel, Real-Time Visual SLAM”. In *Proceedings of the International Conference on Intelligent Robots and Systems*. 2010.
19. Dalal, N. *Finding People in Images and Videos*. Ph.D. Dissertation, Saint Ismier: Institut National Polytechnique de Grenoble, 2006.
20. Dalal, N. and B. Triggs. “Histograms of Oriented Gradients for Human Detection”. *IEEE Computer Society Conference*. 2005.
21. Danker, A. and A. Rosenfeld. “Blob Detection by Relaxation”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1981.
22. Davison, A., Y. G. Cid, and N. Kita. “Real-Time 3D SLAM with Wide-Angle Vision”. In *Proceedings of the Conference on IFAC Symposium on Intelligent Autonomous Vehicles*. July 2004.
23. Dellaert, F. “Assignment 4: RANSAC for Fitting Translations and Homographies”. <http://www.cc.gatech.edu/dellaert/dhtml/home.html>, Oct 2004.
24. Dissanayake, M. W. M. G., P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. “A Solution to the Simultaneous Localization and Map Building (SLAM) Problem”. *IEEE Transactions on Robotics & Automation*, Volume 17, 229–241. 2001.
25. Durrant-Whyte, H. F. and T. Bailey. “Simultaneous Localization and Mapping: Part I”. *IEEE Transactions on Robotics & Automation*, Volume 13, 99–110. 2006.
26. Dynamics, Boston. “LS3 - Legged Squad Support Systems”, 2009. URL http://www.bostondynamics.com/robot_ls3.html.

27. Eliazar, A. and R. Parr. “DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks”. *In Proceedings of the International Conference on Artificial Intelligence*. 2003.
28. Eliazar, A. and R. Parr. “Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps”. *Advances in Neural Information Processing Systems*, 18:339–346, 2006.
29. Fischler, M. A. and R. C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. *Journal on Communications of the ACM*, 24:381–395, 1981.
30. Fontanelli, D., L. Ricciato, and S. Soatto. “A Fast RANSAC Based Registration Algorithm for Accurate Localization in Unknown Environments Using LIDAR Measurements”. *In Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering*. Scottsdale, AZ, 2007.
31. Gionis, A., P. Indyk, and R. Motwani. “Similarity Search in High Dimensions via Hashing”. *In Proceedings of the 25th International Conference on Very Large Data Bases*, 518–529. 1999.
32. Goemans, M. and D. Williamson. “Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming”. *Journal on Communications of the ACM*, 42(6):1115–1145, Nov 1995.
33. Harris, C. and M. Stephens. “A Combined Corner and Edge Detection”. *In Proceedings of the 4th Annual Conference on Alvey Vision*, 147–151. Manchester, UK, 1988.
34. Indyk, P. and R. Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality”. *In Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 604–613. 1998.
35. Jensfelt, J., D. Kragic, J. Folkesson, and M. Bjorkman. “A Framework for Vision Based Bearing Only 3D SLAM”. *In Proceedings of the IEEE International Conference on Robotics and Automation*. Orlando, 2006.
36. Jeong, W. Y. and K. M. Lee. “CV-SLAM: A New Ceiling Vision-Based SLAM Technique”. *In Proceedings of the International Conference on Intelligent Robots and Systems*, 3195–3200. 2005.
37. Kadir, T. and M. Brady. “Scale, Saliency and Image Description”. *International Journal of Computer Vision*, 45(2):83–105, 2001.
38. Kadir, T., A. Zisserman, and M. Brady. “An Affine Invariant Salient Region Detector”. *In Proceedings of the 8th European Conference on Computer Vision*, 345–457. Prague, Czech Republic, 2004.
39. Kaess, M. and F. Dellaert. “Probabilistic Structure Matching for Visual SLAM with a Multi-Camera Rig”. *Journal of Computer Vision and Image Understanding*, 114:286–296, 2010.

40. Karlsson, N., E. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich. "The vSLAM Algorithm for Robust Localization and Mapping". *In Proceedings of the IEEE International Conference on Robotics and Automation*. Barcelona, Spain, Apr 2005.
41. Koenig, S., J. Mitchell, A. Mudgal, and C. Tovey. "A Near-Tight Approximation for the Robot Localization Problem". *Society for Industrial and Applied Mathematics Journal on Computing*, Volume 39, 461–490. 2009.
42. Kulis, B. and K. Grauman. "Kernelized Locality-Sensitive Hashing for Scalable Image Search". *In Proceedings of the 12th IEEE International Conference on Computer Vision*. 2009.
43. Kulis, B., P. Jain, and K. Grauman. "Fast Similarity Search for Learned Metrics". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 31, 2143–2157. 2009.
44. Laviers, K. R. *Concurrent Cognitive Mapping and Localization Using Expected Maximization*. Master's Thesis, Wright-Patterson AFB: Air Force Institute of Technology, 2004.
45. Leon-Garcia, A. *Probability, Statistics, and Random Processes for Electrical Engineering*. Pearson Prentice Hall, 3rd Edition, 2007.
46. Lowe, D. "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision*, 60(2):91–110, 2004.
47. Mikolajczyk, K. and J. Matas. "Improving Descriptors for Fast Tree Matching by Optimal Linear Projection". *In Proceedings of the 11th IEEE International Conference on Computer Vision*, 1–8. 2007.
48. Montiel, J., J. Civera, and A. Davison. "Unified Inverse Depth Parametrization for Monocular SLAM". *In Proceedings of the Conference on Robotics: Science and Systems*. Philadelphia, PA, Aug 2006.
49. Montmerlo, M. and S. Thrun. "Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM". *In Proceedings of the International Conference on Robotics and Automation*, 1985–1991. 2003.
50. Montmerlo, M., S. Thrun, D. Koller, and B. Wegbreit. "Fastslam: A Factored Solution to the Simultaneous Localization and Mapping Problem". *In Proceedings of the Conference on Association for the Advancement of Artificial Intelligence*, 593–598. 2002.
51. Montmerlo, M., S. Thrun, D. Koller, and B. Wegbreit. "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges". *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Acapulco, 2003.
52. Moore, A. W. *An Intoductory Tutorial on kd-Trees*. Technical Report 209, University of Cambridge, 1991.

53. Murphy, K. "Bayesian Map Learning in Dynamic Environments". *In Proceedings of the Conference on Neural Information Processing Systems*. MIT Press, 1999.
54. Naveed, M. *Vision Based Simultaneous Localisation and Mapping for Mobile Robots*. Master's Thesis, Universit de Bourgogne, 2008.
55. Negri, P., X. Clady, M. H. Shehzad, and L. Prevost. "A Cascade of Boosted Generative and Discriminative Classifiers for Vehicle Detection". *European Association for Signal Processing Journal on Advances in Signal Processing*, 2008(136):12, Jan 2008.
56. Pages, J., X. Armangue, J. Salvi, J. Freixenet, and J. Marti. "A Computer Vision System for Autonomous Forklift Vehicles in Industrial Environments". *In Proceedings of the Mediterranean Conference on Control and Automation*. 2001.
57. Pradeep, V., G. Medioni, and J. Weiland. "Visual Loop Closing using Multi-Resolution SIFT Grids in Metric-Topological". *In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1438–1445. Miami, 2009.
58. Riisgaard, S. and M. R. Blas. "SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping", 2003.
59. Saeki, K., K. Tanaka, and T. Ueda. "LSH-RANSAC: An Incremental Scheme for Scalable Localization". *In Proceedings of the International Conference on Robotics and Automation*, 3523 – 3530. Kobe, 2009.
60. Saripalli, S., J. Montgomery, and G. Sukhatme. "Vision-based Autonomous Landing of an Unmanned Aerial Vehicle". *In Proceedings of the IEEE International Conference on Robotics & Automation*, 2799–2804. Washington, DC, 2002.
61. Schempf, H., W. Crowley, C. Gasior, and D. Moreau. "Ultra-Rugged Soldier-Robot for Urban Conflict Missions". *In Proceedings of the Association for Unmanned Vehicle Systems International 30th Annual Symposium and Exhibition Unmanned Systems Conference*. Baltimore, MD, 2003.
62. Scholkopf, B., A. Smola, and KR Muller. "Nonlinear Component Analysis as a Kernel Eigenvalue Problem". *Neural Computation*, 10(5):1299–1319, July 1998.
63. Se, S., D. Lowe, and J. Little. "Global Localization Using Distinctive Visual Features". *In Proceedings of the International Conference on Intelligent Robots and Systems*. EPFL, Lausanne, Switzerland, 2002.
64. Se, S., D. G. Lowe, and J. J. Little. "Vision-based Global Localization and Mapping for Mobile Robots". *IEEE Transactions on Robotics*, Volume 21, 364–375. 2005.
65. Shi, J. and C. Tomasi. "Good Features to Track". *In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 593–600. 1994.

66. Sim, R., P. Elinas, M. Griffin, and J.J. Little. "Vision-Based SLAM Using the Rao-Blackwellised Particle Filter". In *Proceedings of the International Joint Conferences on Artificial Intelligence Workshop Reasoning with Uncertainty in Robotics*. 2005.
67. Sim, R., P. Elinas, M. Griffin, A. Shyr, and J. J. Little. "Design and Analysis of a Framework for Real-Time Vision-based SLAM Using Rao-Blackwellised Particle Filters". In *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision*. 2006.
68. Southall, B., T. Hague, J. A. Merchant, and B. F. Buxton. "Vision-Aided Outdoor Navigation of an Autonomous Horticultural Vehicle". In *Proceedings of the 1st International Conference on Vision Systems*. 2006.
69. Thrun, S. "Robotic Mapping: A Survey". G. Lakemeyer and B. Nebel (Editors), *Exploring Artificial Intelligence in the New Millenium*. Morgan Kauffman, 2002.
70. Thrun, S., W. Burgard, and D. Fox. *Probabalistic Robotics*. Massachusetts Institute of Technology, 2006.
71. Torr, P. *Outlier Detection and Motion Segmentation*. Ph.D. Dissertation, University of Oxford, Engineering Dept., 1995.
72. Torr, P. "Twenty Five Years of RANSAC". In *Proceedings of a Workshop in Conjunction with the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Jun 2006. URL <http://cmp.felk.cvut.cz/ransac-cvpr2006/>.
73. Torr, P. H. S. and D. W. Murray. "Outlier Detection and Motion Segmentation". In *Proceedings of the 6th Conference on SPIE Sensor Fusion*, 432–443. Sep 1993.
74. Trawny, N., A. Mourikis, S. Roumeliotis, A. Johnson, and J. Montgomery. "Vision-Aided Inertial Navigation for Pin-Point Landing Using Observations of Mapped Landmarks". *Journal of Field Robotics*, 24(5):357–378, May 2007.
75. Veth, M. *Fusion of Imaging and Inertial Sensors for Navigation*. Ph.D. Dissertation, Wright-Patterson AFB: Air Force Institute of Technology, 2006.
76. Weyers, C. *Multiple Integrated Navigation Sensors for Improved Occupancy Grid FastSLAM*. Master's Thesis, Wright-Patterson AFB: Air Force Institute of Technology, 2011.
77. Willis, A. and Y. Sui. "Robot 2D Self-Localization Using Range Pattern Matching Via the Discrete Fourier Transform". In *Proceedings of the 2010 IEEE Southeast Conference*, 408–411. Concord, NC, 2010.
78. Won, D. H., S. Chun, S. Sung, T. Kang, and Y. J. Lee. "Improving Mobile Robot Navigation Performance Using Vision based SLAM and Distributed Filters". In *Proceedings of the International Conference on Control, Automation and Systems*. Seoul, Korea, 2008.

79. Wu, A., E. Johnson, and A. Proctor. "Vision-Aided Inertial Navigation for Flight Control". In *Proceedings of the American Institute of Aeronautics and Astronautics Guidance, Navigation, and Control Conference and Exhibit*. 2005.
80. Zuliani, M. "RANSAC for Dummies". <http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf>, Oct 2009.

| | | | | | | | | | | | | |
|---|--------------------|--|---|--|--|---|---|---|---|--------------------------------------|--|--|
| REPORT DOCUMENTATION PAGE | | | | | <i>Form Approved</i> OMB No. 0704-0188 | | | | | | | |
| The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS. | | | | | | | | | | | | |
| 1. REPORT DATE (DD-MM-YYYY) 24-03-2011 | | 2. REPORT TYPE Master's Thesis | | | 3. DATES COVERED (From — To) Sep 2010 — Mar 2011 | | | | | | | |
| 4. TITLE AND SUBTITLE <div style="text-align: center;">Kernelized Locality-Sensitive Hashing for Fast Image Landmark Association Breaks</div> | | | | 5a. CONTRACT NUMBER DACA99-99-C-9999 | | | | | | | | |
| | | | | 5b. GRANT NUMBER | | | | | | | | |
| | | | | 5c. PROGRAM ELEMENT NUMBER | | | | | | | | |
| 6. AUTHOR(S) Mark A. Weems, Capt, USAF | | | | 5d. PROJECT NUMBER 10JON325 | | | | | | | | |
| | | | | 5e. TASK NUMBER | | | | | | | | |
| | | | | 5f. WORK UNIT NUMBER | | | | | | | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765 | | | | | 8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GE/ENG/11-40 | | | | | | | |
| 9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Jacob Campbell, PhD Air Force Research Laboratory, Sensors Directorate, RF Reference Systems AFRL/RYRN 2241 Avionics Circle Wright-Patterson AFB, OH 45433 Jacob.Campbell@wpafb.af.mil (937) 255-6127 x4154 | | | | | 10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RYRN | | | | | | | |
| | | | | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) | | | | | | | |
| 12. DISTRIBUTION / AVAILABILITY STATEMENT This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. | | | | | | | | | | | | |
| 13. SUPPLEMENTARY NOTES | | | | | | | | | | | | |
| 14. ABSTRACT As the concept of war has evolved, navigation in urban environments where GPS may be degraded is increasingly becoming more important. Two existing solutions are vision-aided navigation and vision-based Simultaneous Localization and Mapping (SLAM). The problem, however, is that vision-based navigation techniques can require excessive amounts of memory and increased computational complexity resulting in a decrease in speed. This research focuses on techniques to improve such issues by speeding up and optimizing the data association process in vision-based SLAM. Specifically, this work studies the current methods that algorithms use to associate a current robot pose to that of one previously seen and introduce another method to the image mapping arena for comparison. The current method, <i>kd</i> -trees, is efficient in lower dimensions, but does not narrow the search space enough in higher dimensional datasets. In this research, Kernelized Locality-Sensitive Hashing (KLSH) is implemented to conduct the aforementioned pose associations. Results on KLSH shows that fewer image comparisons are required for location identification than that of other methods. This work can then be extended into a vision-SLAM implementation to subsequently produce a map. | | | | | | | | | | | | |
| 15. SUBJECT TERMS Robotic Mapping; Vision-SLAM, Data Association; KLSH; Nearest Neighbor Search; Artificial Intelligence; Image Processing; Feature Extraction; Mobile Navigation; Vehicle Localization | | | | | | | | | | | | |
| 16. SECURITY CLASSIFICATION OF: <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; padding: 2px;">a. REPORT</td> <td style="width: 33%; padding: 2px;">b. ABSTRACT</td> <td style="width: 33%; padding: 2px;">c. THIS PAGE</td> </tr> <tr> <td style="text-align: center; padding: 2px;">U</td> <td style="text-align: center; padding: 2px;">U</td> <td style="text-align: center; padding: 2px;">U</td> </tr> </table> | | | a. REPORT | b. ABSTRACT | c. THIS PAGE | U | U | U | 17. LIMITATION OF ABSTRACT UU | 18. NUMBER OF PAGES 92 | 19a. NAME OF RESPONSIBLE PERSON Dr. Gilbert L. Peterson, ENG | |
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | | | | | | | | |
| U | U | U | | | | | | | | | | |
| | | | 19b. TELEPHONE NUMBER (include area code) (937) 255-3636, ext 4281; Gilbert.Peterson@afit.edu | | | | | | | | | |